

Solving Large-Scale Robust Stability Problems by Exploiting the Parallel Structure of Polya's Theorem

Reza Kamyar, Matthew M. Peet and Yulia Peet

Abstract—In this paper, we propose a distributed computing approach to solving large-scale robust stability problems on the simplex. Our approach is to formulate the robust stability problem as an optimization problem with polynomial variables and polynomial inequality constraints. We use Polya's theorem to convert the polynomial optimization problem to a set of highly structured Linear Matrix Inequalities (LMIs). We then use a slight modification of a common interior-point primal-dual algorithm to solve the structured LMI constraints. This yields a set of extremely large yet structured computations. We then map the structure of the computations to a decentralized computing environment consisting of independent processing nodes with a structured adjacency matrix. The result is an algorithm which can solve the robust stability problem with the same per-core complexity as the deterministic stability problem with a conservatism which is only a function of the number of processors available. Numerical tests on cluster computers and supercomputers demonstrate the ability of the algorithm to efficiently utilize hundreds and potentially thousands of processors and analyze systems with 100+ dimensional state-space. The proposed algorithms can be extended to perform stability analysis of nonlinear systems and robust controller synthesis.

Index Terms—Robust stability, Polynomial optimization, Large-scale systems, Decentralized computing

I. INTRODUCTION

THIS paper addresses the problem of stability of large-scale systems with several unknown parameters. Control system theory when applied in practical situations often involves the use of large state-space models, typically due to inherent complexity of the system, the interconnection of subsystems, or the reduction of an infinite-dimensional or PDE model to a finite-dimensional approximation. One approach to dealing with such large scale models has been to use model reduction techniques such as balanced truncation [1]. However, the use of model reduction techniques are not necessarily robust and can result in arbitrarily large errors. In addition to large state-space, practical problems often contain uncertainty in the model due to modeling errors, linearization, or fluctuation in the operating conditions. The problem of

stability and control of systems with uncertainty has been widely studied. See, e.g. the texts [2], [3], [4], [5]. However, a limitation of existing computational methods for analysis and control of systems with uncertainty is high complexity. This is a consequence of fact that the problem of robust stability of systems with parametric uncertainty is known to be NP-hard [6], [7]. The result is that for systems with parametric uncertainty and with hundreds of states, existing algorithms will fail with the primary point of failure usually being lack of unallocated memory.

In this paper, we seek to distribute the computation laterally over an array of processors within the context of existing computational resources. Specifically, we seek to utilize cluster-computing, supercomputing and Graphics Processing Unit (GPU)-computing architectures. When designing algorithms to run in a parallel computing environment, one must both synchronize computational tasks among the processors while minimizing communication overhead among the processors. This can be difficult, as each architecture has a specific communication graph. we account for communication by explicitly modeling the required communication graph between processors. This communication graph is then mapped to the processor architecture using the Message-Passing Interface (MPI) [8]. While there are many algorithms for robust stability analysis and control of linear systems, ours is the first which explicitly accounts for the processing architecture in the emerging multi-core computing environment.

Our approach to robust stability is based on the well-established use of parameter-dependent Quadratic-In-The-State (QITS) Lyapunov functions. The use of parameter-dependent Lyapunov QITS functions eliminates the conservativity associated with e.g. quadratic stability [9], at the cost of requiring some restriction on the rate of parameter variation. Specifically, our QITS Lyapunov variables are polynomials in the vector of uncertain parameters. This is a generalization of the use of QITS Lyapunov functions with affine parameter dependence as in [10] and expanded in, e.g. [11], [12], [13], [14]. The use of polynomial QITS Lyapunov variables can be motivated by [15], wherein it is shown that any feasible parameter-dependent LMI with parameters inside a compact set has a polynomial solution or [16] wherein it is shown that local stability of a nonlinear vector field implies the existence of a polynomial Lyapunov function.

There are several results which use polynomial QITS Lyapunov functions to prove robust stability. In most cases, the stability problem is reduced to the general problem of optimization of polynomial variables subject to LMI constraints - an NP-hard problem [17]. To avoid NP-hardness, the poly-

Manuscript received November 21, 2011; revised September 24, 2012; accepted February 26, 2013. This work was supported entirely by funding from the National Science Foundation under Award CMMI-1100376. Recommended by XXXX.

R. Kamyar is with the Department of Mechanical Engineering, Cybernetic Systems and Controls Laboratory, Arizona State University, Tempe, AZ 85281 USA (e-mail: rkamyar@asu.edu).

M. M. Peet and Y. Peet are with the School for Engineering of Matter, Transport, and Energy, Engineering Research Center, Arizona State University, Tempe, AZ 85281 USA (e-mail: mpeet@asu.edu; ypeet@asu.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TAC.2013.2253253

mial optimization problem is usually solved in an asymptotic manner by posing a sequence of sufficient conditions of increasing accuracy and decreasing conservatism. For example, building on the result in [15], [18] provides a sequence of increasingly precise LMI conditions for robust stability analysis of linear systems with affine dependency on uncertain parameters on the complex unit ball. Necessary and sufficient stability conditions for linear systems with one uncertain parameter are derived in [19], providing an explicit bound on the degree of the polynomial-type Lyapunov function. The result is extended to multi-parameter-dependent linear systems in [20]. Another important approach to optimization of polynomials is the Sum of Squares (SOS) methodology which replaces the polynomial positivity constraint with the constraint that the polynomial admits a representation as a sum of squares of polynomials [21], [22], [23], [24]. A version of this theorem for polynomials with matrix coefficients can be found in [23]. While we have worked extensively with the SOS methodology, we have not, as of yet, been able to adapt algorithms for solving the resulting LMI conditions to a parallel-computing environment. Finally, there have been several results in recent years on the use of Polyá's theorem to solve polynomial optimization problems [25] on the simplex. An extension of the Polyá's theorem for uncertain parameters on the multisimplex or hypercube can be found in [26]. The approach presented in this paper is an extension of the use of Polyá's theorem for solving polynomial optimization problems in a parallel computing environment.

The goal of this project is to create algorithms which explicitly map computation, communication and storage to existing parallel processing architectures. This goal is motivated by the failure of existing general-purpose Semi-Definite Programming (SDP) solvers to efficiently utilize platforms for large-scale computation. Specifically, it is well-established that linear programming and semi-definite programming both belong to the complexity class P-Complete, also known as the class of inherently sequential problems. Although there have been several attempts to map certain SDP solvers to a parallel computing environment [27], [28], certain critical steps cannot be distributed. The result is that as the number of processors increases, certain bottleneck computations dominate, leading a saturation in computational speed of these solvers (Amdahl's law [29]). We avoid these bottleneck computations and communications by exploiting the particular structure of the LMI conditions associated with Polyá's theorem. Note that, in principle, a perfectly designed general-purpose SDP algorithm could identify the structure of the SDP, as we have, and map the communication, computation and memory constraints to the parallel architecture. Indeed, there has been a great deal of research on creating programming languages which attempt to do just this [30], [31]. However, at present such languages are mostly theoretical and have certainly not been incorporated into existing SDP solvers.

In addition to parallel SDP solvers, there have been some efforts to exploit structure in certain polynomial optimization algorithms to reducing the size and complexity of the resulting LMI's. For example, in [32] symmetry was used to reduce the size of the SDP variables. Specific sparsity structure

was used in [33], [34], [35] to reduce the complexity of the linear algebra calculations. Generalized approaches to the use of sparsity in SDP algorithms can be found in [34]. Groebner basis techniques [36], [37] have been used by [33] to simplify the formulation of the SDPs associated with the SOS decomposition problems.

The paper is organized around two independent problems: setting up the sequence of structured SDPs associated with Polyá's theorem and solving them. Note that the problem of decentralizing the set-up algorithm is significant in that for large-scale systems, the instantiation of the problem may be beyond the memory and computational capacity of a single processing node. For the set-up problem, the algorithm that we propose has no centralized memory or computational requirements whatsoever. Furthermore, if a sufficient number of processors are available, the number of messages does not change with the size of the state-space or the number of Polyá's iterations. In addition, the ideal communication architecture for the set-up algorithm does not correspond to the communication structure of GPU computing or supercomputing. In the second problem, we propose a variant of a standard SDP primal-dual algorithm and map the computational, memory and communication requirements to a parallel computing environment. Unlike the set-up algorithm, the primal-dual algorithm does have a small centralized component corresponding to the update of the set of dual variables. However, we have structured the algorithm so that the size of this dual computation is solely a function of the degree of the polynomial QITS Lyapunov function and does not depend on the number of Polyá's iterations, meaning that the sequence of algorithms has fixed centralized computational and communication complexity. In addition, there is no communication between processors, which means that the algorithm is well suited to most parallel computing architectures. A graph representation of the communication architecture of both the set-up and SDP algorithms has also been provided in the relevant sections.

Combining the set-up and SDP components and testing the result of both in cluster computing environments, we demonstrate the capability of robust analysis and control of systems with 100+ states and several uncertain parameters. Specifically, we ran a series of numerical experiments using a local Linux cluster and the Blue Gene supercomputer (with 200 processor allocation). First, we applied the algorithm to a current problem in robust stability analysis of magnetic confinement fusion using a discretized PDE model. Next, we examine the accuracy of the algorithm as Polyá's iterations progress and compare this accuracy with the SOS approach. We show that unlike the general-purpose parallel SDP solver SDPARA [28], the speed-up - the increase in processing speed per additional processor - of our algorithm shows no evidence of saturation. Finally, we calculate the envelope of the algorithm on the Linux cluster in terms of the maximum state-space dimension, number of processors and Polyá's iterations.

NOTATION

We represent l -variate monomials as $\alpha^\gamma = \prod_{i=1}^l \alpha_i^{\gamma_i}$, where $\alpha \in \mathbb{R}^l$ is the vector of variables and $\gamma \in \mathbb{N}^l$ is the

vector of exponents and $\sum_{i=1}^l \gamma_i = d$ is the degree of the monomial. We define $W_d := \left\{ \gamma \in \mathbb{N}^l : \sum_{i=1}^l \gamma_i = d \right\}$ as the totally ordered set of the exponents of l -variate monomials of degree d , where the ordering is lexicographic. In lexicographical ordering $\gamma \in W_d$ precedes $\eta \in W_d$, if the left most non-zero entry of $\gamma - \eta$ is positive. The lexicographical index of every $\gamma \in W_d$ can be calculated using the map $\langle \cdot \rangle : \mathbb{N}^l \rightarrow \mathbb{N}$ defined as [38]

$$\langle \gamma \rangle = \sum_{j=1}^{l-1} \sum_{i=1}^{\gamma_i} f\left(l-j, d+1 - \sum_{k=1}^{j-1} \gamma_k - i\right) + 1, \quad (1)$$

where as in [39]

$$f(l, d) := \begin{cases} 0 & \text{for } l = 0 \\ \binom{l+d-1}{l-1} = \frac{(d+l-1)!}{d!(l-1)!} & \text{for } l > 0, \end{cases} \quad (2)$$

is the cardinality of W_d , i.e., the number of l -variate monomials of degree d . For convenience, we also define the index of a monomial α^γ to be $\langle \gamma \rangle$. We represent l -variate homogeneous polynomials of degree d_p as

$$P(\alpha) = \sum_{\gamma \in W_{d_p}} P_{\langle \gamma \rangle} \alpha^\gamma, \quad (3)$$

where $P_{\langle \gamma \rangle} \in \mathbb{R}^{n \times n}$ is the matrix coefficient of the monomial α^γ . We denote the element corresponding to the i^{th} row and j^{th} column of matrix A as $[A]_{i,j}$. The subspace of symmetric matrices in $\mathbb{R}^{n \times n}$ is denoted by \mathbb{S}^n . We define a basis for \mathbb{S}^n as

$$[E_k]_{i,j} := \begin{cases} 1 & \text{if } i = j = k \\ 0 & \text{otherwise} \end{cases}, \quad \text{for } k \leq n \quad \text{and} \\ [E_k]_{i,j} := [F_k]_{i,j} + [F_k]_{i,j}^T, \quad \text{for } k > n, \quad (4)$$

where

$$[F_k]_{i,j} := \begin{cases} 1 & \text{if } i = j - 1 = k - n \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Note that this choice of basis is arbitrary - any other basis could be used. However, any change in basis would require modifications to the formulae defined in this paper. The canonical basis for \mathbb{R}^n is denoted by e_i for $i = 1, \dots, n$,

where $e_i = [0 \dots 0 \overbrace{1}^{i^{\text{th}}} 0 \dots 0]$. The vector with all entries equal to one is denoted by $\vec{1}$. The trace of $A \in \mathbb{R}^{n \times n}$ is denoted by $\text{tr}(A) = \sum_{i=1}^n [A]_{i,i}$. The block-diagonal matrix with diagonal blocks $X_1, \dots, X_m \in \mathbb{R}^{n \times n}$ is denoted $\text{diag}(X_1, \dots, X_m) \in \mathbb{R}^{mn \times mn}$ or occasionally as $\text{diag}(X_i)_{i=1}^m \in \mathbb{R}^{mn \times mn}$. The identity and zero matrices are denoted by $I_n \in \mathbb{R}^{n \times n}$ and $0_n \in \mathbb{R}^{n \times n}$.

II. PRELIMINARIES

Consider the linear system

$$\dot{x}(t) = A(\alpha)x(t), \quad (6)$$

where $A(\alpha) \in \mathbb{R}^{n \times n}$ and $\alpha \in Q \subset \mathbb{R}^l$ is a vector of uncertain parameters. In this paper, we consider the case where $A(\alpha)$

is a homogeneous polynomial and $Q = \Delta_l \subset \mathbb{R}^l$ where Δ_l is the unit simplex, i.e.,

$$\Delta_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^l \alpha_i = 1, \alpha_i \geq 0 \right\}. \quad (7)$$

If $A(\alpha)$ is not homogeneous, we can obtain an equivalent homogeneous representation in the following manner. Suppose $A(\alpha)$ is a non-homogeneous polynomial with $\alpha \in \Delta_l$, is of degree d_a and has N_a monomials with non-zero coefficients. Define $D = (d_{a_1}, \dots, d_{a_{N_a}})$, where d_{a_i} is the degree of the i^{th} monomial of $A(\alpha)$ according to lexicographical ordering. Now define the polynomial $B(\alpha)$ as per the following.

- 1) Let $B = A$.
- 2) For $i = 1, \dots, N_a$, multiply the i^{th} monomial of $B(\alpha)$, according to lexicographical ordering, by $\left(\sum_{j=1}^l \alpha_j\right)^{d_a - d_{a_i}}$.

Then, since $\sum_{j=1}^l \alpha_j = 1$, $B(\alpha) = A(\alpha)$ for all $\alpha \in \Delta_l$ and hence all properties of $\dot{x}(t) = A(\alpha)x(t)$ are retained by the homogeneous system $\dot{x}(t) = B(\alpha)x(t)$.

1) *Example: Construction of the homogeneous system*
 $\dot{x}(t) = B(\alpha)x(t)$.

Consider the non-homogeneous polynomial $A(\alpha) = C\alpha_1^2 + D\alpha_2 + E\alpha_3 + F$ of degree $d_a = 2$, where $[\alpha_1, \alpha_2, \alpha_3] \in \Delta_3$. Using the above procedure, the homogeneous polynomial $B(\alpha)$ can be constructed as

$$\begin{aligned} B(\alpha) &= C\alpha_1^2 + D\alpha_2(\alpha_1 + \alpha_2 + \alpha_3) + E\alpha_3(\alpha_1 + \alpha_2 + \alpha_3) \\ &+ F(\alpha_1 + \alpha_2 + \alpha_3)^2 = \underbrace{(C+F)\alpha_1^2}_{B_1} + \underbrace{(D+2F)\alpha_1\alpha_2}_{B_2} \\ &+ \underbrace{(E+2F)\alpha_1\alpha_3}_{B_3} + \underbrace{(D+F)\alpha_2^2}_{B_4} + \underbrace{(D+E+2F)\alpha_2\alpha_3}_{B_5} \\ &+ \underbrace{(E+F)\alpha_3^2}_{B_6} = \sum_{\gamma \in W_2} B_{\langle \gamma \rangle} \alpha^\gamma. \end{aligned} \quad (8)$$

The following is a stability condition [25] for System (6).

Theorem 1: System (6) is stable if and only if there exists a polynomial matrix $P(\alpha)$ such that $P(\alpha) \succ 0$ and

$$A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0 \quad (9)$$

for all $\alpha \in \Delta_l$.

A similar condition also holds for discrete-time linear systems. The conditions associated with Theorem 1 are infinite-dimensional LMIs, meaning they must hold at infinite number of points. Such problems are known to be NP-hard [17]. In this paper we derive a sequence of polynomial-time algorithms such that their outputs converge to the solution of the infinite-dimensional LMI. Key to this result is Polya's Theorem [40]. A variation of this theorem for matrices is given as follows.

Theorem 2: (Polya's Theorem) The homogeneous polynomial $F(\alpha) \succ 0$ for all $\alpha \in \Delta_l$ if and only if for all sufficiently large d ,

$$\left(\sum_{i=1}^l \alpha_i \right)^d F(\alpha) \quad (10)$$

has all positive definite coefficients.

Upper bounds for Polyá's exponent d can be found as in [41]. However, these bounds are based on the properties of F and are difficult to determine a priori. In this paper, we show that applying Polyá's Theorem to the robust stability problem, i.e., the inequalities in Theorem 1 yields a semi-definite programming condition with an efficiently distributable structure. This is discussed in the following section.

III. PROBLEM SET-UP

In this section, we show how Polyá's theorem can be used to determine the robust stability of an uncertain system using linear matrix inequalities with a distributable structure.

A. Polyá's Algorithm

We consider the stability of the system described by Equation (6). We are interested in finding a $P(\alpha)$ which satisfies the conditions of Theorem 1. According to Polyá's theorem, the constraints of Theorem 1 are satisfied if for some sufficiently large d_1 and d_2 , the polynomials

$$\left(\sum_{i=1}^l \alpha_i \right)^{d_1} P(\alpha) \quad \text{and} \quad (11)$$

$$- \left(\sum_{i=1}^l \alpha_i \right)^{d_2} (A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha)) \quad (12)$$

have all positive definite coefficients.

Let $P(\alpha)$ be a homogeneous polynomial of degree d_p which can be represented as

$$P(\alpha) = \sum_{\gamma \in W_{d_p}} P_{\langle \gamma \rangle} \alpha^\gamma, \quad (13)$$

where the coefficients $P_{\langle \gamma \rangle} \in \mathbb{S}^n$ and where we recall that $W_{d_p} := \left\{ \gamma \in \mathbb{N}^l : \sum_{i=1}^l \gamma_i = d_p \right\}$ is the set of the exponents of all l -variate monomials of degree d_p . Since $A(\alpha)$ is a homogeneous polynomial of degree d_a , we can write it as

$$A(\alpha) = \sum_{\gamma \in W_{d_a}} A_{\langle \gamma \rangle} \alpha^\gamma, \quad (14)$$

where the coefficients $A_{\langle \gamma \rangle} \in \mathbb{R}^{n \times n}$. By substituting (13) and (14) into (11) and (12) and defining d_{pa} as the degree of $P(\alpha)A(\alpha)$, the conditions of Theorem 2 can be represented in the form

$$\sum_{h \in W_{d_p}} \beta_{\langle h \rangle, \langle \gamma \rangle} P_{\langle h \rangle} \succ 0; \quad \gamma \in W_{d_p+d_1} \quad \text{and} \quad (15)$$

$$\sum_{h \in W_{d_p}} (H_{\langle h \rangle, \langle \gamma \rangle}^T P_{\langle h \rangle} + P_{\langle h \rangle} H_{\langle h \rangle, \langle \gamma \rangle}) \prec 0; \quad \gamma \in W_{d_{pa}+d_2}. \quad (16)$$

Here $\beta_{\langle h \rangle, \langle \gamma \rangle}$ is defined to be the scalar coefficient which multiplies $P_{\langle h \rangle}$ in the $\langle \gamma \rangle$ -th monomial of the homogeneous polynomial $\left(\sum_{i=1}^l \alpha_i \right)^{d_1} P(\alpha)$ using the lexicographical ordering. Likewise $H_{\langle h \rangle, \langle \gamma \rangle} \in \mathbb{R}^{n \times n}$ is the term which left or right multiplies $P_{\langle h \rangle}$ in the $\langle \gamma \rangle$ -th monomial of $\left(\sum_{i=1}^l \alpha_i \right)^{d_2} (A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha))$ using the lexicographical ordering. For an intuitive explanation as to how these

β and H terms are calculated, we consider a simple example. Precise formulae for these terms will follow the example.

1) Example: Calculating the β and H coefficients.

Consider $A(\alpha) = A_1 \alpha_1 + A_2 \alpha_2$ and $P(\alpha) = P_1 \alpha_1 + P_2 \alpha_2$. By expanding Equation (11) for $d_1 = 1$ we have

$$(\alpha_1 + \alpha_2)P(\alpha) = P_1 \alpha_1^2 + (P_1 + P_2) \alpha_1 \alpha_2 + P_2 \alpha_2^2. \quad (17)$$

The $\beta_{\langle h \rangle, \langle \gamma \rangle}$ terms are then extracted as

$$\beta_{1,1} = 1, \quad \beta_{2,1} = 0, \quad \beta_{1,2} = 1, \quad \beta_{2,2} = 1, \quad \beta_{1,3} = 0, \quad \beta_{2,3} = 1. \quad (18)$$

Next, by expanding Equation (12) for $d_2 = 1$ we have

$$\begin{aligned} (\alpha_1 + \alpha_2) (A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha)) &= (A_1^T P_1 + P_1 A_1) \alpha_1^3 \\ &+ (A_1^T P_1 + P_1 A_1 + A_2^T P_1 + P_1 A_2 + A_1^T P_2 + P_2 A_1) \alpha_1^2 \alpha_2 \\ &+ (A_2^T P_1 + P_1 A_2 + A_1^T P_2 + P_2 A_1 + A_2^T P_2 + P_2 A_2) \alpha_1 \alpha_2^2 \\ &+ (A_2^T P_2 + P_2 A_2) \alpha_2^3. \end{aligned} \quad (19)$$

The $H_{\langle h \rangle, \langle \gamma \rangle}$ terms are then extracted as

$$\begin{aligned} H_{1,1} &= A_1, \quad H_{2,1} = \mathbf{0}, \quad H_{1,2} = A_1 + A_2, \quad H_{2,2} = A_1, \\ H_{1,3} &= A_2, \quad H_{2,3} = A_1 + A_2, \quad H_{1,4} = \mathbf{0}, \quad H_{2,4} = A_2. \end{aligned} \quad (20)$$

2) *General Formula:* The $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$ can be formally defined recursively as follows. Let the initial values for $\beta_{\langle h \rangle, \langle \gamma \rangle}$ be defined as

$$\beta_{\langle h \rangle, \langle \gamma \rangle}^{(0)} = \begin{cases} 1 & \text{if } h = \gamma \\ 0 & \text{otherwise} \end{cases} \quad \text{for } \gamma \in W_{d_p} \text{ and } h \in W_{d_p}. \quad (21)$$

Then, iterating for $i = 1, \dots, d_1$, we let

$$\beta_{\langle h \rangle, \langle \gamma \rangle}^{(i)} = \sum_{\lambda \in W_1} \beta_{\langle h \rangle, \langle \gamma - \lambda \rangle}^{(i-1)} \quad \text{for } \gamma \in W_{d_p+i} \text{ and } h \in W_{d_p}. \quad (22)$$

Finally, we set $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\} = \{\beta_{\langle h \rangle, \langle \gamma \rangle}^{d_1}\}$. To obtain $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$, set the initial values as

$$H_{\langle h \rangle, \langle \gamma \rangle}^{(0)} = \sum_{\lambda \in W_{d_a}: \lambda + h = \gamma} A_{\langle \lambda \rangle} \quad \text{for } \gamma \in W_{d_p+d_a} \text{ and } h \in W_{d_p}. \quad (23)$$

Then, iterating for $i = 1, \dots, d_2$, we let

$$H_{\langle h \rangle, \langle \gamma \rangle}^{(i)} = \sum_{\lambda \in W_1} H_{\langle h \rangle, \langle \gamma - \lambda \rangle}^{(i-1)} \quad \text{for } \gamma \in W_{d_{pa}+i} \text{ and } h \in W_{d_p}. \quad (24)$$

Finally, set $\{H_{\langle h \rangle, \langle \gamma \rangle}\} = \{H_{\langle h \rangle, \langle \gamma \rangle}^{d_2}\}$.

For the case of large-scale systems, computing and storing $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$ and $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$ is a significant challenge due to the number of these coefficients. Specifically, the number of terms increases with l (number of uncertain parameters in system (6)), d_p (degree of $P(\alpha)$), d_{pa} (degree of $P(\alpha)A(\alpha)$) and d_1, d_2 (Polyá's exponents) as follows.

3) *Number of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ coefficients:* For given l, d_p and d_1 , since $h \in W_{d_p}$ and $\gamma \in W_{d_p+d_1}$, the number of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ coefficients is the product of $L_0 := \text{card}(W_{d_p})$ and $L := \text{card}(W_{d_p+d_1})$. Recall that $\text{card}(W_{d_p})$ is the number of all l -variate monomials of degree d_p and can be calculated using (2)

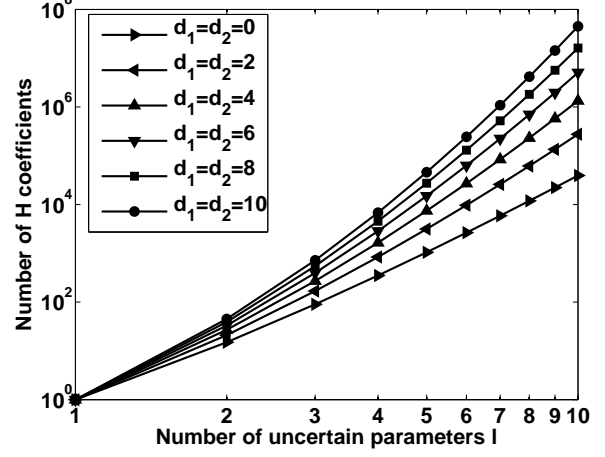
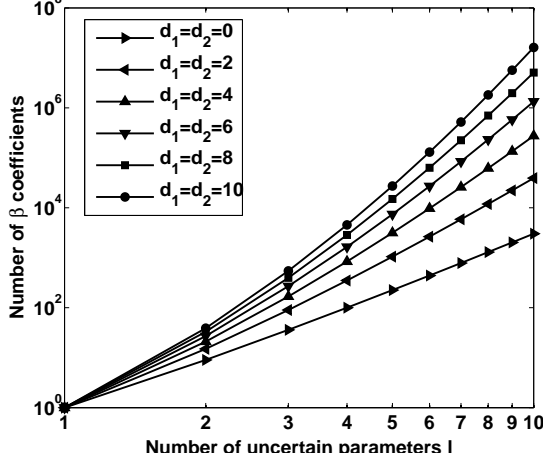


Fig. 1. Number of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients vs. the number of uncertain parameters for different Polya's exponents and for $d_p = d_a = 2$

as follows.

$$L_0 = f(l, d_p) = \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_p + l - 1}{l - 1} = \frac{(d_p + l - 1)!}{d_p! (l - 1)!} & \text{for } l > 0. \end{cases} \quad (25)$$

Likewise, $\text{card}(W_{d_p+d_1})$, i.e., the number of all l -variate monomials of degree $d_p + d_1$ is calculated using (2) as follows.

$$L = f(l, d_p + d_1) = \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_p + d_1 + l - 1}{l - 1} = \frac{(d_p + d_1 + l - 1)!}{(d_p + d_1)! (l - 1)!} & \text{for } l > 0. \end{cases} \quad (26)$$

The number of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ coefficients is $L_0 \cdot L$.

4) *Number of $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients:* For given l, d_p, d_a and d_2 , since $h \in W_{d_p}$ and $\gamma \in W_{d_{pa}+d_2}$, the number of $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients is the product of $L_0 := \text{card}(W_{d_p})$ and $M := \text{card}(W_{d_{pa}+d_2})$. By using (2), we have

$$M = f(l, d_{pa} + d_2) = \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_{pa} + d_2 + l - 1}{l - 1} = \frac{(d_{pa} + d_2 + l - 1)!}{(d_{pa} + d_2)! (l - 1)!} & \text{for } l > 0. \end{cases} \quad (27)$$

The number of $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients is $L_0 \cdot M$.

The number of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients and the required memory to store these coefficients are shown in Figs. 1 and 2 in terms of the number of uncertain parameters l and for different Polya's exponents. In all cases $d_p = d_a = 2$.

It is observed from Fig. 2 that, even for small d_p and d_a , the required memory is in the Terabyte range. In [38], we proposed a decentralized computing approach to the calculation of $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$ on large cluster computers. In the present work, we extend this method to the calculation of $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$ and the SDP elements which will be discussed in the following section. We express the LMIs associated with conditions (15) and (16) as an SDP in both primal and dual forms. We also discuss the structure of the primal and dual SDP variables and the constraints.

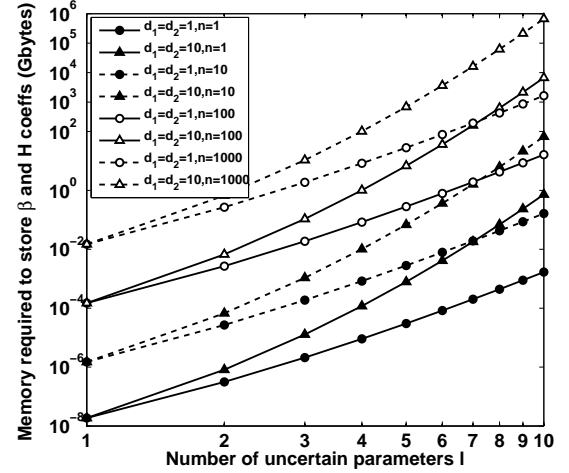


Fig. 2. Memory required to store β and H coefficients vs. number of uncertain parameters, for different d_1, d_2 and $d_p = d_a = 2$

B. SDP Problem Elements

A semi-definite programming problem can be stated either in primal or dual format. Given $C \in \mathbb{S}^m$, $a \in \mathbb{R}^K$ and $B_i \in \mathbb{S}^m$, the **primal problem** is of the form

$$\begin{aligned} \max_X \quad & \text{tr}(CX) \\ \text{subject to} \quad & a - B(X) = 0 \\ & X \succeq 0, \end{aligned} \quad (28)$$

where the linear operator $B : \mathbb{S}^m \rightarrow \mathbb{R}^K$ is defined as

$$B(X) = [\text{tr}(B_1 X) \quad \text{tr}(B_2 X) \quad \cdots \quad \text{tr}(B_K X)]^T. \quad (29)$$

$X \in \mathbb{S}^m$ is the primal variable. Given a primal SDP, the associated **dual problem** is

$$\begin{aligned} \min_{y, Z} \quad & a^T y \\ \text{subject to} \quad & B^T(y) - C = Z \\ & Z \succeq 0, \quad y \in \mathbb{R}^K, \end{aligned} \quad (30)$$

where $B^T : \mathbb{R}^K \rightarrow \mathbb{S}^m$ is the transpose operator and is given by

$$B^T(y) = \sum_{i=1}^K y_i B_i \quad (31)$$

and where $y \in \mathbb{R}^K$ and $Z \in \mathbb{S}^m$ are the dual variables. The elements C , B_i and a of the SDP problem associated with the LMIs in (15) and (16) are defined as follows. We define the element C as

$$C := \text{diag}(C_1, \dots, C_L, C_{L+1}, \dots, C_{L+M}), \quad (32)$$

where

$$C_i := \begin{cases} \delta I_n \cdot \left(\sum_{h \in W_{d_p}} \beta_{\langle h \rangle, i} \frac{d_p!}{h_1! \dots h_l!} \right), & 1 \leq i \leq L \\ 0_n, & L+1 \leq i \leq L+M, \end{cases} \quad (33)$$

where recall that $L = \text{card}(W_{d_p+d_1})$ is the number of monomials in $\left(\sum_{i=1}^l \alpha_i \right)^{d_1} P(\alpha)$, $M = \text{card}(W_{d_{pa}+d_2})$ is the number of monomials in $\left(\sum_{i=1}^l \alpha_i \right)^{d_2} P(\alpha)A(\alpha)$, where n is the dimension of system (6), l is the number of uncertain parameters and δ is a small positive parameter.

For $i = 1, \dots, K$, define B_i elements as

$$B_i := \text{diag}(B_{i,1}, \dots, B_{i,L}, B_{i,L+1}, \dots, B_{i,L+M}), \quad (34)$$

where K is the number of dual variables in (30) and is equal to the product of the number of upper-triangular elements in each $P_\gamma \in \mathbb{S}^n$ (the coefficients in $P(\alpha)$) and the number of coefficients in $P(\alpha)$ (i.e. the cardinality of W_{d_p}). Since there are $f(l, d_p) = \binom{d_p + l - 1}{l - 1}$ coefficients in $P(\alpha)$ and each coefficient has $\tilde{N} := \frac{1}{2}n(n+1)$ upper-triangular elements, we find

$$K = \frac{(d_p + l - 1)!}{d_p!(l - 1)!} \tilde{N}. \quad (35)$$

To define the $B_{i,j}$ blocks, first we define the function $V_{\langle h \rangle} : \mathbb{Z}^K \rightarrow \mathbb{Z}^{n \times n}$,

$$V_{\langle h \rangle}(x) := \sum_{j=1}^{\tilde{N}} E_j x_{j+\tilde{N}(\langle h \rangle-1)} \quad \text{for all } h \in W_{d_p}, \quad (36)$$

which maps each variable to a basis matrix E_j , where recall that E_j is the basis for \mathbb{S}^n . Note that a different choice of basis would require a different function $V_{\langle h \rangle}$. Then for $i = 1, \dots, K$,

$$B_{i,j} := \begin{cases} \sum_{h \in W_{d_p}} \beta_{\langle h \rangle, j} V_{\langle h \rangle}(e_i), & 1 \leq j \leq L \text{ (I)} \\ -\sum_{h \in W_{d_p}} \left(H_{\langle h \rangle, j-L}^T V_{\langle h \rangle}(e_i) + V_{\langle h \rangle}(e_i) H_{\langle h \rangle, j-L} \right), & L+1 \leq j \leq L+M. \text{ (II)} \end{cases} \quad (37)$$

Finally, to complete the SDP problem associated with Polya's algorithm set

$$a = \vec{1} \in \mathbb{R}^K. \quad (38)$$

C. Parallel Set-up Algorithm

In this section, we propose a decentralized, iterative algorithm for calculating the terms $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$, $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$, C and B_i as defined in (22), (24), (32) and (34). The algorithm has been implemented in C++, using MPI (Message Passing Interface) and is available at: www.sites.google.com/a/asu.edu/kamyar/software. We present an abridged description of this algorithm in Algorithm 1, wherein N is the number of available processors.

Note that we have only addressed the problem of robust stability analysis, using the polynomial inequality

$$P(\alpha) \succ 0, A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0$$

for $\alpha \in \Delta_l$. However, we can generalize the decentralized set-up algorithm to consider a more general class of feasibility problems, i.e.,

$$\sum_{i=1}^{\hat{N}} \left(\tilde{A}_i(\alpha) \tilde{X}(\alpha) \tilde{B}_i(\alpha) + \tilde{B}_i^T(\alpha) \tilde{X}(\alpha) \tilde{A}_i^T(\alpha) + R_i(\alpha) \right) \prec 0 \quad (41)$$

for $\alpha \in \Delta_l$. One motivation behind the development of such generalized set-up algorithm is that the parameter-dependent versions of the LMIs associated with H_2 and H_∞ synthesis problems in [42], [43] can be formulated in the form of (41).

D. Set-up algorithm: Complexity Analysis

Since checking the positive definiteness of all representatives of a square matrix with parameters on proper real intervals is intractable [7], the question of feasibility of (9) is also intractable. To solve the problem of inherent intractability we establish a trade off between accuracy and complexity. In fact, we develop a sequence of decentralized polynomial-time algorithms whose solutions converge to the exact solution of the NP-hard problem. In other words, the translation of a polynomial optimization problem to an LMI problem is the main source of complexity. This high complexity is unavoidable and, in fact, is the reason we seek parallel algorithms.

Algorithm 1 distributes the computation and storage of $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$ and $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$ among the processors and their dedicated memories, respectively. In an ideal case, where the number of available processors is sufficiently large (equal to the number of monomials in $P(\alpha)A(\alpha)$, i.e. M) only one monomial (L_0 of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and L_0 of $H_{\langle h \rangle, \langle \gamma \rangle}$) are assigned to each processor.

1) *Computational complexity analysis*: The most computationally expensive part of the set-up algorithm is the calculation of the $B_{i,j}$ blocks in (37). Considering that the cost of matrix-matrix multiplication is $\sim n^3$, the cost of calculating each $B_{i,j}$ block is $\sim \text{card}(W_{d_p}) \cdot n^3$. According to (34) and (37), the total number of $B_{i,j}$ blocks is $K(L+M)$. Hence, as per Algorithm 1, each processor processes $K(\text{floor}(\frac{L}{N}) + \text{floor}(\frac{M}{N}))$ of the $B_{i,j}$ blocks, where N is the number of available processors. Thus the per processor computational cost of calculating the $B_{i,j}$ at each Polya's iteration is

$$\sim \text{card}(W_{d_p}) \cdot n^3 \cdot K \left(\text{floor} \left(\frac{L}{N} \right) + \text{floor} \left(\frac{M}{N} \right) \right). \quad (42)$$

Algorithm 1: The parallel set-up algorithm

Inputs: d_p : degree of $P(\alpha)$, d_a : degree of $A(\alpha)$, n : number of states, l : number of uncertain parameters, d_1, d_2 : number of Polya's iterations, Coefficients of $A(\alpha)$.

Initialization: Set $\hat{d}_1 = \hat{d}_2 = 0$ and $d_{pa} = d_p + d_a$. Calculate L_0 as the number of monomials in $P(\alpha)$ using (25) and M as the number of monomials in $P(\alpha)A(\alpha)$ using (27). Set $L = L_0$. Calculate $L' = \text{floor}(\frac{L}{N})$ and $M' = \text{floor}(\frac{M}{N})$ as the number of monomials in $P(\alpha)$ and $P(\alpha)A(\alpha)$ assigned to each processor.

for $i = 1, \dots, N$, *processor* i **do**
 Initialize $\beta_{k,j}$ for $j = (i-1)L' + 1, \dots, iL'$ and $k = 1, \dots, L_0$ using (21).
 Initialize $H_{k,m}$ for $m = (i-1)M' + 1, \dots, iM'$ and $k = 1, \dots, L_0$ using (23).

Calculating β and H coefficients:

while $\hat{d}_1 \leq d_1$ or $\hat{d}_2 \leq d_2$ **do**
 if $\hat{d}_1 \leq d_1$ **then**
 for $i = 1, \dots, N$, *processor* i **do**
 Set $d_p = d_p + 1$ and $\hat{d}_1 = \hat{d}_1 + 1$. Update L using (26). Update $L' = \text{floor}(\frac{L}{N})$.
 Calculate $\beta_{k,j}$ for $j = (i-1)L' + 1, \dots, iL'$ and $k = 1, \dots, L_0$ using (22).
 if $\hat{d}_2 \leq d_2$ **then**
 for $i = 1, \dots, N$, *processor* i **do**
 Set $d_{pa} = d_{pa} + 1$ and $\hat{d}_2 = \hat{d}_2 + 1$. Update M using (27). Update $M' = \text{floor}(\frac{M}{N})$.
 Calculate $H_{k,m}$ for $m = (i-1)M' + 1, \dots, iM'$ and $k = 1, \dots, L_0$ using (24).

Calculating the SDP elements:

for $i = 1, \dots, N$, *processor* i **do**
 Calculate the number of dual variables K using (35).
 Set $T' = \text{floor}(\frac{L+M}{N})$.
 Calculate the blocks of the SDP element C as

$$\begin{cases} C_j \text{ using (33)} & \text{for } j = (i-1)L' + 1, \dots, iL' \\ C_j = 0_n & \text{for } j = L + (i-1)M' + 1, \dots, L + iM' \end{cases}$$

Set the sub-blocks of the SDP element C as

$$\bar{C}_i = \text{diag} \left(C_k |_{k=(i-1)T'+1}^{iT'} \right). \quad (39)$$

for $j = 1, \dots, K$ **do**

 Calculate the blocks of the SDP elements B_j as

$$\begin{cases} B_{j,k} \text{ using (37)-I} & \text{for } k = (i-1)L' + 1, \dots, iL' \\ B_{j,k} \text{ using (37)-II} & \text{for } k = L + (i-1)M' + 1, \dots, L + iM' \end{cases}$$

 Set the sub-blocks of the SDP element B_j as

$$\bar{B}_{j,i} = \text{diag} \left(B_{j,k} |_{k=(i-1)T'+1}^{iT'} \right). \quad (40)$$

Outputs: Sub-blocks \bar{C}_i and $\bar{B}_{j,i}$ of the SDP elements for $i = 1, \dots, N$ and $j = 1, \dots, K$.

By substituting for K from (35), $\text{card}(W_{d_p})$ from (25), L from (26) and M from (27), the per processor computation cost at each iteration is

$$\sim \left(\frac{(d_p + l - 1)!}{d_p!(l-1)!} \right)^2 \frac{n^4}{2(n+1)} \left(\text{floor} \left(\frac{(d_p + d_1 + l - 1)!}{(d_p + d_1)!(l-1)!} \frac{1}{N} \right) + \text{floor} \left(\frac{(d_{pa} + d_2 + l - 1)!}{(d_{pa} + d_2)!(l-1)!} \frac{1}{N} \right) \right) \quad (43)$$

assuming that $l > 0$ and $N \leq M$. For example, for the case of large-scale systems (large n and l), the computation cost per processor at each iteration is $\sim (l^{2d_p+d_1} + l^{2d_p+d_a+d_2})n^5$ having $N = L_0$ processors, $\sim (l^{2d_p+d_1} + l^{2d_p+d_a+d_2})n^5$ having $N = L$ processors and $\sim l^{2d_p+d_a+d_2-d_1}n^5$ having $N = M$ processors. Thus for the case where $d_p \geq 3$, the number of operations grows more slowly in n than in l .

2) *Communication complexity analysis:* Communication between processors can be modeled by a directed graph $G(V, E)$, where the set of nodes $V = \{1, \dots, N\}$ is the set of indices of the available processors and the set of edges $E = \{(i, j) : i, j \in V\}$ is the set of all pairs of processors that communicate with each other. For every directed graph we can define an adjacency matrix T_G . If processor i communicates with processor j , then $[T_G]_{i,j} = 1$, otherwise $[T_G]_{i,j} = 0$. In this section, we only define the adjacency matrix for the part of the algorithm that performs Polya's iterations on $P(\alpha)$. For Polya's iterations on $P(\alpha)A(\alpha)$, the adjacency matrix can be defined in a similar manner. For simplicity, we assume that at each iteration, the number of available processors is equal to the number of monomials in $(\sum_{i=1}^l \alpha_i)^{d_1} P(\alpha)$. Using (26), let us define r_{d_1} and r_{d_1+1} as the numbers of monomials in $(\sum_{i=1}^l \alpha_i)^{d_1} P(\alpha)$ and $(\sum_{i=1}^l \alpha_i)^{d_1+1} P(\alpha)$. For $I = 1, \dots, r_{d_1}$, define

$$\mathcal{E}_I := \{\text{lex. indices of monomials in } \left(\sum_{i=1}^l \alpha_i \right) \alpha^\gamma : \gamma \in W_{d_p+d_1} \text{ and } \langle \gamma \rangle = I\}.$$

Then for $i = 1, \dots, r_{d_1+1}$ and $j = 1, \dots, r_{d_1+1}$,

$$[T_G]_{i,j} := \begin{cases} 1 & \text{if } i \leq r_{d_1} \text{ and } j \in \mathcal{E}_i \text{ and } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

Note that this definition implies that the communication graph of the set-up algorithm changes at every iteration. To help visualize the graph, the adjacency matrix for the case where

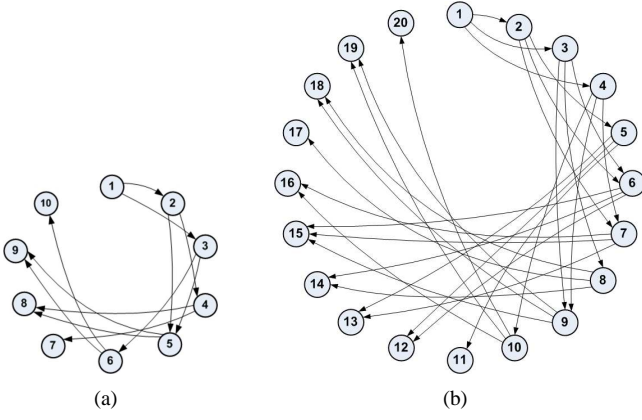


Fig. 3. Graph representation of the network communication of the set-up algorithm. (a) Communication directed graph for the case $\alpha \in \Delta_3, d_p = 2$. (b) Communication directed graph for the case $\alpha \in \Delta_4, d_p = 2$.

$\alpha \in \Delta_2$ is

$$T_G := \left[\begin{array}{cccccc|cccc} 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \hline 0 & \cdots & \cdots & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & 0 & \cdots & 0 \end{array} \right] \in \mathbb{R}^{r_{d_1+1} \times r_{d_1+1}},$$

where the nonzero sub-block of T_G lies in $\mathbb{R}^{r_{d_1} \times r_{d_1}}$. We can also illustrate the communication graphs for the cases $\alpha \in \Delta_3$ and $\alpha \in \Delta_4$ with $d_p = 2$ as seen in Fig. 3(a) and 3(b).

For a given algorithm, the communication complexity is defined as the sum of the size of all communicated messages. For simplicity, let us consider the worst case scenario, where each processor is assigned more than one monomial and sends all of its assigned $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients to other processors. In this case, the algorithm assigns $\text{floor}(\frac{L}{N}) \cdot \text{card}(W_{d_p})$ of the $\beta_{\langle h \rangle, \langle \gamma \rangle}$ coefficients, each of size 1, and $(\text{floor}(\frac{L}{N}) + \text{floor}(\frac{M}{N})) \cdot \text{card}(W_{d_p})$ of the $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients, each of size n^2 , to each processor. Thus the communication complexity of the algorithm per processor and per iteration is

$$\text{card}(W_{d_p}) \left(\text{floor} \left(\frac{L}{N} \right) + \text{floor} \left(\frac{M}{N} \right) n^2 \right). \quad (44)$$

This indicates that increasing the number of processors (up to M) actually leads to less communication overhead per processor and improves the scalability of the algorithm. By substituting for $\text{card}(W_{d_p})$ from (25), L from (26) and M from (27) and considering large l and n , the communication complexity per processor at each Poly's iteration is $\sim l^{d_{pa}+d_2} n^2$ having $N = L_0$ processors, $\sim l^{d_{pa}+d_2-d_1} n^2$ having $N = L$ processors and $\sim l^{d_p} n^2$ having $N = M$ processors.

IV. PARALLEL SDP SOLVER

In this section, we describe the steps of our primal-dual interior-point algorithm and show how, for the LMIs in (15)

and (16), these steps can be distributed in a distributed-computing, distributed-memory environment.

A. Interior-point methods

Interior-point methods define a popular class of algorithms for solving linear and semi-definite programming problems. The most widely accepted interior-point algorithms are dual scaling [44], [45], primal-dual [46], [47], [48] and cutting-plane/spectral bundle [49], [50], [51]. In this paper, we use the central-path-following primal-dual algorithm described in [48] and [27]. Although we found it possible to use dual-scaling algorithms, we chose to pursue a primal-dual algorithm because, in general, primal-dual algorithms converge faster [48], [45] while still preserving the structure of the solution (see (59)) at each iteration. We prefer primal-dual to cutting plane/spectral bundle methods because, as we show in Section IV-D, the centralized part of our primal-dual algorithm consists of solving a symmetric system of linear equations (see (80)), whereas for the cutting plane/spectral bundle algorithm, the centralized computation would consist of solving a constrained quadratic program (see [50], [51]) with number of variables equal to the size of the system of linear equations. Because centralized computation is the limiting factor in a parallel algorithm, and because solving symmetric linear equations is simpler than solving a quadratic programming problem, we chose the primal-dual approach.

The choice of a central path-following primal-dual algorithm as in [48] and [52] was motivated by results in [53] which demonstrated better convergence, accuracy and robustness over the other types of primal-dual algorithms. More specifically, we chose the approach in [48] over [52] because unlike the Schur complement matrix (SCM) approach of the algorithm in [52], the SCM of [48] is symmetric and only the upper-triangular elements need to be sent/received by the processors. This leads to less communication overhead. The other reason for choosing [48] is that the symmetric SCM of the algorithm in [48] can be factorized using Cholesky factorization, whereas the non-symmetric SCM of [52] must be factorized by LU factorization (LU factorization is roughly twice as expensive as Cholesky factorization). Since factorization of SCM comprises the main portion of centralized computation in our algorithm, it is crucial for us to use computationally cheaper factorization methods to achieve better scalability.

In the primal-dual algorithm, both primal and dual problems are solved by iteratively calculating primal and dual step directions and step sizes, and applying these to the primal and dual variables. Let X be the primal variable and y and Z be the dual variables. At each iteration, the variables are updated as

$$X_{k+1} = X_k + t_p \Delta X \quad (45)$$

$$y_{k+1} = y_k + t_d \Delta y \quad (46)$$

$$Z_{k+1} = Z_k + t_d \Delta Z, \quad (47)$$

where ΔX , Δy , and ΔZ are Newton's search direction and t_p and t_d are primal and dual step sizes. We choose the step sizes using a standard line-search between 0 and 1 with the

constraint that X_{k+1} and Z_{k+1} remain positive semi-definite. We use a Newton's search direction given by

$$\Delta X = \Delta \hat{X} + \Delta \bar{X} \quad (48)$$

$$\Delta y = \Delta \hat{y} + \Delta \bar{y} \quad (49)$$

$$\Delta Z = \Delta \hat{Z} + \Delta \bar{Z}, \quad (50)$$

where $\Delta \hat{X}$, $\Delta \hat{y}$ and $\Delta \hat{Z}$ are the predictor step directions and $\Delta \bar{X}$, $\Delta \bar{y}$, and $\Delta \bar{Z}$ are the corrector step directions. As per [48], the predictor step directions are found as

$$\Delta \hat{y} = \Omega^{-1}(-a + B(Z^{-1}GX)) \quad (51)$$

$$\Delta \hat{X} = -X + Z^{-1}GB^T(\Delta \hat{y})X \quad (52)$$

$$\Delta \hat{Z} = B^T(y) - Z - C + B^T(\Delta \hat{y}), \quad (53)$$

where C and the operators B and B^T are as defined in the previous section,

$$G = -B^T(y) + Z + C, \quad (54)$$

and

$$\Omega = [B(Z^{-1}B^T(e_1)X) \cdots B(Z^{-1}B^T(e_k)X)]. \quad (55)$$

Recall that e_1, \dots, e_k are the standard basis for \mathbb{R}^k . Once we have the predictor step directions, we can calculate the corrector step directions as per [48]. Let $\mu = \frac{1}{3}tr(ZX)$. The corrector step directions are

$$\Delta \bar{y} = \Omega^{-1}(B(\mu Z^{-1}) - B(Z^{-1}\Delta \hat{Z}\Delta \hat{X})) \quad (56)$$

$$\Delta \bar{X} = \mu Z^{-1} - Z^{-1}\Delta \hat{Z}\Delta \hat{X} - Z^{-1}\Delta \bar{Z}X \quad (57)$$

$$\Delta \bar{Z} = B^T(\Delta \bar{y}). \quad (58)$$

The stopping criterion is $|a^T y - tr(CX)| \leq \epsilon$. Information regarding the selection of starting points and convergence of different variants of interior-point primal-dual algorithm, including the algorithm we use in this paper are presented in [46], [47] and [48].

B. Structure of SDP Variables

The key algorithmic insight of this paper which allows us to use the primal-dual approach presented in [48] is that by choosing an initial value for the primal variable with a certain block structure corresponding to the distributed structure of the processors, the algorithm will preserve this structure at every iteration. Specifically, we define the following structured block-diagonal subspace where each block corresponds to a single processor.

$$S_{l,m,n} := \{Y \in \mathbb{R}^{(l+m)n \times (l+m)n} : Y = \text{diag}(Y_1, \dots, Y_l, Y_{l+1}, \dots, Y_{l+m}) \text{ for } Y_i \in \mathbb{R}^{n \times n}\} \quad (59)$$

According to the following theorem, the subspace $S_{l,m,n}$ is invariant under Newton's iteration in the sense that when the algorithm in [48] is applied to the SDP problem defined by the polynomial optimization problem with initial value of the primal variable $X_0 \in S_{l,m,n}$, then the primal variable remains in the subspace at every Newton's iteration X_k .

Theorem 3: Consider the SDP problem defined in (28) and (30) with elements given by (32), (34) and (38). Suppose

L and M are the cardinalities of $W_{d_p+d_1}$ and $W_{d_{pa}+d_2}$. If (45), (46) and (47) are initialized by

$$X_0 \in S_{L,M,n}, \quad y_0 \in \mathbb{R}^K, \quad Z_0 \in S_{L,M,n}, \quad (60)$$

then for all $k \in \mathbb{N}$,

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}. \quad (61)$$

Proof: We proceed by induction. First, suppose for some $k \in \mathbb{N}$,

$$X_k \in S_{L,M,n} \quad \text{and} \quad Z_k \in S_{L,M,n}. \quad (62)$$

We would like to show that this implies $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. To see this, observe that according to (45)

$$X_{k+1} = X_k + t_p \Delta X_k \quad \text{for all } k \in \mathbb{N}. \quad (63)$$

From (48), ΔX_k can be written as

$$\Delta X_k = \Delta \hat{X}_k + \Delta \bar{X}_k \quad \text{for all } k \in \mathbb{N}. \quad (64)$$

To find the structure of ΔX_k , we focus on the structures of $\Delta \hat{X}_k$ and $\Delta \bar{X}_k$ individually. Using (52), $\Delta \hat{X}_k$ is

$$\Delta \hat{X}_k = -X_k + Z_k^{-1}G_k B^T(\Delta \hat{y}_k)X_k \quad \text{for all } k \in \mathbb{N}. \quad (65)$$

where according to (54), G_k is

$$G_k = C - B^T(y_k) + Z_k \quad \text{for all } k \in \mathbb{N}. \quad (66)$$

First we examine the structure of G_k . According to the definition of C and B_i in (32) and (34), and the definition of $B^T(y)$ in (31), we know that

$$C \in S_{L,M,n}, \quad B^T : \mathbb{R}^K \mapsto S_{L,M,n}. \quad (67)$$

Since all the terms on the right hand side of (66) are in $S_{L,M,n}$ and $S_{L,M,n}$ is a subspace, we conclude

$$G_k \in S_{L,M,n}. \quad (68)$$

Returning to (65), using our assumption in (62) and noting that the structure of the matrices in $S_{L,M,n}$ is also preserved through multiplication and inversion, we conclude

$$\Delta \hat{X}_k \in S_{L,M,n}. \quad (69)$$

Using (57), the second term in (64) is

$$\Delta \bar{X}_k = \mu Z_k^{-1} - Z_k^{-1}\Delta \hat{Z}_k \Delta \hat{X}_k - Z_k^{-1}\Delta \bar{Z}_k X_k \quad \text{for all } k \in \mathbb{N}. \quad (70)$$

To determine the structure of $\Delta \bar{X}_k$, first we investigate the structure of $\Delta \hat{Z}_k$ and $\Delta \bar{Z}_k$. According to (53) and (58) we have

$$\Delta \hat{Z}_k = B^T(y_k) - Z_k - C + B^T(\Delta \hat{y}_k) \quad \text{for all } k \in \mathbb{N} \quad (71)$$

$$\Delta \bar{Z}_k = B^T(\Delta \bar{y}_k) \quad \text{for all } k \in \mathbb{N}. \quad (72)$$

Since all the terms in the right hand side of (71) and (72) are in $S_{L,M,n}$, then

$$\Delta \hat{Z}_k \in S_{L,M,n}, \quad \Delta \bar{Z}_k \in S_{L,M,n}. \quad (73)$$

Recalling (69), (70) and our assumption in (62), we have

$$\Delta \bar{X}_k \in S_{L,M,n}. \quad (74)$$

According to (69), (73) and (74), the total step directions are in $S_{L,M,n}$,

$$\Delta X_k = \Delta \hat{X}_k + \Delta \bar{X}_k \in S_{L,M,n} \quad (75)$$

$$\Delta Z_k = \Delta \hat{Z}_k + \Delta \bar{Z}_k \in S_{L,M,n}, \quad (76)$$

and it follows that

$$X_{k+1} = X_k + t_p \Delta X_k \in S_{L,M,n} \quad (77)$$

$$Z_{k+1} = Z_k + t_p \Delta Z_k \in S_{L,M,n}. \quad (78)$$

Thus, for any $y \in \mathbb{R}^K$ and $k \in \mathbb{N}$, if $X_k, Z_k \in S_{L,M,n}$, we have $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. Since we have assumed that the initial values $X_0, Z_0 \in S_{L,M,n}$, we conclude by induction that $X_k \in S_{L,M,n}$ and $Z_k \in S_{L,M,n}$ for all $k \in \mathbb{N}$. ■

C. Parallel Implementation

In this section, a parallel algorithm for solving the SDP problems associated with Polya's algorithm is provided. We show how to map the block-diagonal structure of the primal variable and Newton updates described in Section IV-A to a parallel computing structure consisting of a central root processor with N slave processors. Note that processor steps are simultaneous and transitions between root and processor steps are synchronous. Processors are idle when root is active and vice-versa. A C++ implementation of this algorithm, using MPI and Blas/Lapack libraries is provided at: www.sites.google.com/a/asu.edu/kamyar/software. Let N be the number of available processors and $J = \text{floor}(\frac{L+M}{N})$. As per Algorithm 1, we assume processor i has access to the sub-blocks $\bar{\mathbf{C}}_i$ and $\bar{\mathbf{B}}_{j,i}$ defined in (39) and (40) for $j = 1, \dots, K$. Be aware that minor parts of Algorithm 2 have been abridged in order to simplify the presentation.

D. Computational Complexity Analysis: SDP Algorithm

$\text{NC} \subset \text{P}$ is defined to be the class of problems which can be solved in a poly-logarithmic number of steps using a polynomially number processors and is often considered to be the class of problems that can be parallelized efficiently. The class P-complete is a set of problems which are equivalent up to an NC reduction, but contains no problem in NC and is thought to be the simplest class of "inherently sequential" problems. It has been proven that Linear Programming (LP) is P-complete [54] and SDP is P-hard (at least as hard as any P-complete problem) and thus is unlikely to admit a general-purpose parallel solution. Given this fact and given the observation that the problem we are trying to solve is NP-hard, it is important to thoroughly understand the complexity of the algorithms we are proposing and how this complexity scales with various parameters which define the size of the problem. To better understand these issues, we have broken our complexity analysis down into several cases which should be of interest to the control community. Note that the cases below do not discuss memory complexity. This is because in the cases when a sufficient number of processors are available, for a system with n states, the memory requirements per block are simply proportional to n^2 .

1) Case 1: Systems with large number of states

Algorithm 2: The parallel SDP solver algorithm

Inputs: $\bar{\mathbf{C}}_i, \bar{\mathbf{B}}_{j,i}$ for $i = 1, \dots, N$ and $j = 1, \dots, K$ - the sub-blocks of the SDP elements provided to processor i by the set-up algorithm.

Processors Initialization step:

for $i = 1, \dots, N$, processor i **do**

Initialize primal and dual variables $\mathbf{X}_i^0, \mathbf{Z}_i^0$ and y^0 as

$$\mathbf{X}_i^0 = \begin{cases} I_{(J+1)n}, & 0 \leq i < L + M - NJ \\ I_{Jn}, & L + M - NJ \leq i < N, \end{cases}$$

$$\mathbf{Z}_i^0 = \mathbf{X}_i^0 \quad \text{and} \quad y^0 = \vec{0} \in \mathbb{R}^K,$$

Calculate the complementary slackness [49]

$S_i = \text{tr}(\mathbf{Z}_i^0 \mathbf{X}_i^0)$. Send S_i to processor root.

Root Initialization step:

Root processor **do**

Calculate the barrier parameter [49] $\mu = \frac{1}{3} \sum_{i=1}^N S_i$.

Set the SDP element $a = \vec{1} \in \mathbb{R}^K$.

Processors step 1:

for $i = 1, \dots, N$, processor i **do**

for $k = 1, \dots, K$ **do**

Calculate the elements of Ω_1 (R-H-S of (80))

$$\omega_{i,k} = \text{tr} \left(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1} \left(- \sum_{j=1}^K y_j \bar{\mathbf{B}}_{j,i} + \mathbf{Z}_i + \bar{\mathbf{C}}_i \right) \mathbf{X}_i \right)$$

for $l = 1, \dots, K$ **do**

Calculate the elements of the SCM as

$$\lambda_{i,k,l} = \text{tr} \left(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1} \bar{\mathbf{B}}_{l,i} \mathbf{X}_i \right) \quad (79)$$

Send $\omega_{i,k}$ and $\lambda_{i,k,l}$, $k = 1, \dots, K$ and $l = 1, \dots, K$ to root processor.

Root step 1:

Root processor **do**

Construct the R-H-S of (80) and the SCM as

$$\Omega_1 = \begin{pmatrix} \sum_{i=1}^N \omega_{i,1} \\ \sum_{i=1}^N \omega_{i,2} \\ \vdots \\ \sum_{i=1}^N \omega_{i,K} \end{pmatrix} - a \quad \text{and}$$

$$\Lambda = \left[\begin{pmatrix} \sum_{i=1}^N \lambda_{i,1,1} \\ \sum_{i=1}^N \lambda_{i,2,1} \\ \vdots \\ \sum_{i=1}^N \lambda_{i,K,1} \end{pmatrix}, \dots, \begin{pmatrix} \sum_{i=1}^N \lambda_{i,1,K} \\ \sum_{i=1}^N \lambda_{i,2,K} \\ \vdots \\ \sum_{i=1}^N \lambda_{i,K,K} \end{pmatrix} \right]$$

Solve the following system of equations for the predictor dual step $\Delta \hat{y} \in \mathbb{R}^K$ and send $\Delta \hat{y}$ to all processors.

$$\Lambda \Delta \hat{y} = \Omega_1 \quad (80)$$

Processors step 2:**for** $i = 1, \dots, N$, processor i **do**

Calculate the predictor step directions

$$\begin{aligned}\Delta \hat{\mathbf{X}}_i &= -\mathbf{X}_i \\ &+ (\mathbf{Z}_i)^{-1} \left(-\sum_{j=1}^K y_j \bar{\mathbf{B}}_{j,i} + \mathbf{Z}_i + \bar{\mathbf{C}}_i \right) \sum_{j=1}^K \Delta \hat{y}_j \bar{\mathbf{B}}_{j,i} \mathbf{X}_i, \\ \Delta \hat{\mathbf{Z}}_i &= \sum_{j=1}^K y_j \bar{\mathbf{B}}_{j,i} - \mathbf{Z}_i - \bar{\mathbf{C}}_i + \sum_{j=1}^K \Delta \hat{y}_j \bar{\mathbf{B}}_{j,i}.\end{aligned}$$

for $k = 1, \dots, K$ **do**Calculate the elements of Ω_2 (R-H-S of (81))

$$\delta_{i,k} = \text{tr}(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}), \tau_{i,k} = \text{tr}(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1} \Delta \hat{\mathbf{Z}}_i \Delta \hat{\mathbf{X}}_i)$$

Send $\delta_{i,k}$ and $\tau_{i,k}$, $k = 1, \dots, K$ to root processor.**Root step 2:**Root processor **do**

Construct the R-H-S of (81) as

$$\begin{aligned}\Omega_2 &= \mu \left[\sum_{i=1}^N \delta_{i,1} \quad \sum_{i=1}^N \delta_{i,2} \quad \cdots \quad \sum_{i=1}^N \delta_{i,K} \right]^T - \\ &\quad \left[\sum_{i=1}^N \tau_{i,1} \quad \sum_{i=1}^N \tau_{i,2} \quad \cdots \quad \sum_{i=1}^N \tau_{i,K} \right]^T\end{aligned}$$

Solve the following system of equations for the corrector dual variable $\Delta \bar{y}$ and send $\Delta \bar{y}$ to all processors.

$$\Lambda \Delta \bar{y} = \Omega_2 \quad (81)$$

Processors step 3:**for** $i = 1, \dots, N$, processor i **do**

Calculate the corrector step directions as follows.

$$\Delta \bar{\mathbf{Z}}_i = \sum_{j=1}^K \Delta \bar{y}_j \bar{\mathbf{B}}_{j,i}$$

$$\Delta \bar{\mathbf{X}}_i = -(\mathbf{Z}_i)^{-1} (\Delta \bar{\mathbf{Z}}_i \mathbf{X}_i + \Delta \hat{\mathbf{Z}}_i \Delta \hat{\mathbf{X}}_i) + \mu (\mathbf{Z}_i)^{-1}$$

Calculate primal dual step total step directions as follows.

$$\Delta \mathbf{X}_i = \Delta \hat{\mathbf{X}}_i + \Delta \bar{\mathbf{X}}_i, \Delta \mathbf{Z}_i = \Delta \hat{\mathbf{Z}}_i + \Delta \bar{\mathbf{Z}}_i, \Delta y = \Delta \hat{y} + \Delta \bar{y}.$$

Set primal step size t_p and dual step size t_d using an appropriate line search methos.

Update primal and dual variables as

$$\mathbf{X}_i \equiv \mathbf{X}_i + t_p \Delta \mathbf{X}_i, \quad \mathbf{Z}_i \equiv \mathbf{Z}_i + t_d \Delta \mathbf{Z}_i, \quad y \equiv y + t_d \Delta y$$

Processors step 4:**for** $i = 1, \dots, N$, processor i **do**

Calculate the contribution to primal cost

 $\tilde{\phi}_i = \text{tr}(\bar{\mathbf{C}}_i \mathbf{X}_i)$ and the complementary slack $S_i = \text{tr}(\mathbf{Z}_i \mathbf{X}_i)$. Send S_i and $\tilde{\phi}_i$ to root processor.**Root step 4:**Root processor **do**

Update the barrier parameter $\mu = \frac{1}{3} \sum_{i=1}^N S_i$. Calculate primal and dual costs as $\phi = \sum_{i=1}^N \tilde{\phi}_i$ and $\psi = a^T y$. If $|\phi - \psi| > \varepsilon$, then go to Processors step 1; Otherwise calculate the coefficients of $P(\alpha)$ as $P_i = \sum_{j=1}^N E_j y_{(j+\tilde{N}-1)}$ for $i = 1, \dots, L_0$.

Suppose we are considering a problem with n states. For this case, the most expensive part of the algorithm is the calculation of the Schur complement matrix Λ by the processors in Processors step 1 (and summed by the root in Root step 1, although we neglect this part). In particular, the computational complexity of the algorithm is determined by the number of operations required to calculate (79), restated here.

$$\lambda_{i,k,l} = \text{tr}(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1} \bar{\mathbf{B}}_{l,i} \mathbf{X}_i) \quad (82)$$

for $k = 1, \dots, K$ and $l = 1, \dots, K$.

Since the cost of $n \times n$ matrix-matrix multiplication is $\sim n^3$ and each of $\mathbf{X}_i, \mathbf{Z}_i, \bar{\mathbf{B}}_{l,i}$ has $\text{floor}(\frac{L+M}{N})$ number of blocks in $\mathbb{R}^{n \times n}$, the number of operations performed by the i^{th} processor to calculate $\lambda_{i,k,l}$ for $k = 1, \dots, K$ and $l = 1, \dots, K$ is

$$\begin{cases} \sim \text{floor}\left(\frac{L+M}{N}\right) K^2 n^3 & N < L+M \\ \sim K^2 n^3 & N \geq L+M \end{cases} \quad (83)$$

at each iteration, where $i = 1, \dots, N$. By substituting K in (83) from (35), for $N \geq L+M$, each processor performs

$$\sim \frac{((d_p + l - 1)!)^2}{(d_p!)^2 ((l - 1)!)^2} n^7 \quad (84)$$

operations per iteration. Therefore, for systems with large n and fixed d_p and l , the number of operations per processor required to solve the SDP associated with parameter-dependent feasibility problem $A(\alpha)^T P(\alpha) + P(\alpha) A(\alpha) \prec 0$, is proportional to n^7 . Solving the LMI associated with the parameter-independent problem $A^T P + P A \prec 0$ using our algorithm or most of the SDP solvers such as [55], [27], [28] also requires $O(n^7)$ operations per processor. Therefore, if we have a sufficient number of processors, the proposed algorithm solves both the stability and robust stability problems by performing $O(n^7)$ operations per processor in this case.

2) Case 2: High Accuracy/Low Conservativity

In this case we consider the effect of raising Polya's exponent. Consider the definition of simplex as follows.

$$\tilde{\Delta}_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^l \alpha_i = r, \alpha_i \geq 0 \right\} \quad (85)$$

Suppose we now define the accuracy of the algorithm as the largest value of r found by the algorithm (if it exists) such that if the uncertain parameters lie inside the corresponding simplex, the stability of the system is verified. Typically, increasing Polya's exponent d in (10) improves the accuracy of the algorithm. If we again only consider Processor step 1, according to (84), the number of processor operations is independent of the Polya's exponent d_1 and d_2 ! Because this part of the algorithm does not vary with Polya's exponent, we look at the root processing requirements associated with solving the systems of equations in (80) and (81) in Root step 1 using Cholesky factorization. Each of these systems consists of K equations. The computational complexity of Cholesky factorization is $O(K^3)$. Thus the number of operations performed by the root processor is proportional to

$$K^3 = \frac{((d_p + l - 1)!)^3}{(d_p!)^3 ((l - 1)!)^3} n^6. \quad (86)$$

In terms of communication complexity, the most significant operation between the root and other processors is sending and receiving $\lambda_{i,k,l}$ for $i = 1, \dots, N$, $k = 1, \dots, K$ and $l = 1, \dots, K$ in Processors step 1 and Root step 1. Thus the total communication cost for N processors per iteration is

$$\sim N \cdot K^2 = N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l - 1)!)^2} n^4. \quad (87)$$

From (84), (86) and (87) it is observed that the number of processors operations, root operations and communication operations are independent of Polya's exponent d_1 and d_2 . Therefore, we conclude that for a fixed d_p and sufficiently large number of processors N ($N \geq L + M$), improving the accuracy by increasing d_1 and d_2 does not add any computation per processor or communication overhead.

3) Case 3: Algorithm scalability/Speed-up

The speed-up of a parallel algorithm is defined as $SP_N = \frac{T_s}{T_N}$, where T_s is the execution time of the sequential algorithm and T_N is the execution time of the parallel algorithm using N processors. The speed-up is governed by

$$SP_N = \frac{N}{D + NS}, \quad (88)$$

where D is defined as the ratio of the total operations performed by all processors except root to total operations performed by all processors and root. S is the ratio of the operations performed by root to total operations performed by all processors and root. Suppose that the number of available processors is equal to the number of sub-blocks in C defined in (32). Using the above definitions for D and S , Equation (84) as the decentralized computation and (86) as the centralized computation, D and S can be approximated as

$$D \simeq \frac{N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l - 1)!)^2} n^7}{N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l - 1)!)^2} n^7 + \frac{((d_p + l - 1)!)^3}{(d_p!)^3((l - 1)!)^3} n^6} \quad \text{and} \quad (89)$$

$$S \simeq \frac{\frac{((d_p + l - 1)!)^3}{(d_p!)^3((l - 1)!)^3} n^6}{N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l - 1)!)^2} n^7 + \frac{((d_p + l - 1)!)^3}{(d_p!)^3((l - 1)!)^3} n^6}. \quad (90)$$

According to (26) and (27) the number of processors $N = L + M$ is independent of n ; Therefore

$$\lim_{n \rightarrow \infty} D = 1 \quad \text{and} \quad \lim_{n \rightarrow \infty} S = 0.$$

By substituting D and S in (88) with their limit values, we have $\lim_{n \rightarrow \infty} SP_N = N$. Thus, for large n , by using $L + M$ processors the presented decentralized algorithm solves large robust stability problems $L + M$ times faster than the sequential algorithms. For different values of the state-space dimension n , the theoretical speed-up of the algorithm versus the number of processors is illustrated in Fig. 4. As shown in Fig. 4, for problems with large n , by using $N \leq L + M$ processors the parallel algorithm solves the robust stability problems approximately N times faster than the sequential algorithm. As n increases, the trend of speed-up becomes increasingly linear. Therefore, in case of problems with a large number of

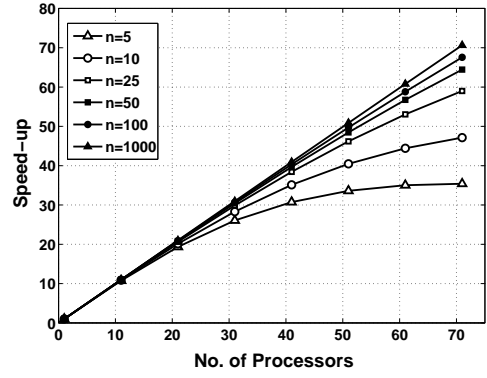


Fig. 4. Theoretical speed-up vs. No. of processors for different system dimensions n for $l = 10$, $d_p = 2$, $d_a = 3$ and $d_1 = d_2 = 4$, where $L + M = 53625$

states n , our algorithm becomes increasingly efficient in terms of processor utilization.

4) Case 4: Synchronization and load balancing

The proposed algorithm is synchronous in that all processors must return values before the centralized step can proceed. However, in the case where we have fewer processors than blocks, some processors may be assigned one block more than other processors. In this case, some processors may remain idle while waiting for the more heavily loaded blocks to complete. In the worst case, this can result in a 50% decrease in speed. We have addressed this issue in the following manner:

- 1) We allocate almost the same number (± 1) of blocks of the SDP elements C and B_i to all processors, i.e., $\text{floor}(\frac{L+M}{N}) + 1$ blocks to r processors and $\text{floor}(\frac{L+M}{N})$ blocks to the other $N - r$ processors, where r is the remainder of dividing $L + M$ by N .
- 2) We assign the same routine to all of the processors in the Processors steps of Alg. 2.

If $L + M$ is a multiple of N , then the algorithm assigns the same amount of data, i.e., $\frac{L+M}{N}$ blocks of C and B_i to each processor. In this case, the processors are perfectly synchronized. If $L + M$ is not a multiple of N , then according to (83), r of N processors perform $K^2 n^3$ extra operations per iteration. This fraction is $\frac{1}{1 + \text{floor}(\frac{L+M}{N})} \leq 0.5$ of the operations per iteration performed by each of r processors. Thus in the worst case, we have a 50% reduction, although this situation is rare. As an example, the load balancing (distribution of data and calculation) for the case of solving an SDP of the size $L + M = 24$ using different numbers of available processors N is demonstrated in Fig. 5. This figure shows the number of blocks that are allocated to each processor. According to this figure, for $N = 2, 12$ and 24 , the processors are well-balanced, whereas for the case where $N = 18$, twelve processors perform 50% fewer calculations.

5) Case 5: Communication graph

The communication directed graph of the SDP algorithm (Fig. 6) is static (fixed for all iterations). At each iteration, root sends messages ($\Delta \hat{y}$ and $\Delta \bar{y}$) to all of the processors and receives messages ($\lambda_{i,k,l}$ in (79)) from all of the processors. The adjacency matrix of the communication directed graph is

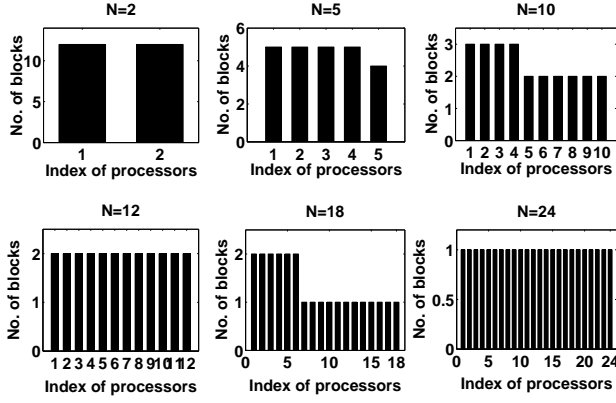


Fig. 5. The number of blocks of the SDP elements assigned to each processor; An illustration of load balancing

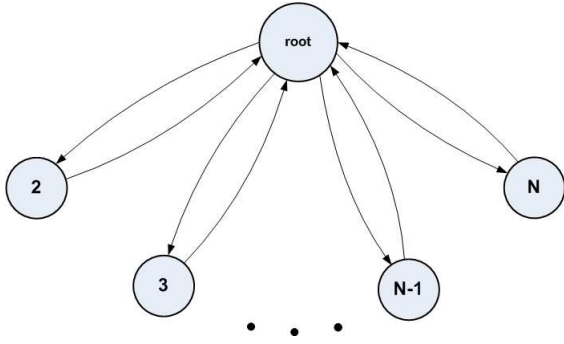


Fig. 6. The communication graph of the SDP algorithm

defined as follows. For $i = 1, \dots, N$ and $j = 1, \dots, N$,

$$[T_G]_{i,j} := \begin{cases} 1 & \text{if } (i = 1 \text{ or } j = 1) \text{ and } (i \neq j) \\ 0 & \text{Otherwise.} \end{cases}$$

V. TESTING AND VALIDATION

In this section, we present validation data in 4 key areas. First, we present analysis results for a realistic large-scale model of tokamak operation using a discretized PDE model. Next we present accuracy and convergence data and compare our algorithm to the SOS approach. Next, we analyze scalability and speed-up of our algorithm as we increase the number of processors and compare our results to the general-purpose parallel SDP solver SDPARA. Finally, we explore the limits of the algorithm in terms of problems size when implemented on a moderately powerful cluster computer and using a moderate processor allocation on the Blue Gene supercomputer.

1) *Example 1: Application to control of a discretized PDE model in fusion research.*

The goal of this example is to use the proposed algorithm to solve a real-world stability problem. A simplified model for the poloidal magnetic flux gradient in a Tokamak reactor [56] is

$$\frac{\partial \psi_x(x, t)}{\partial t} = \frac{1}{\mu_0 a^2} \frac{\partial}{\partial x} \left(\frac{\eta(x)}{x} \frac{\partial}{\partial x} (x \psi_x(x, t)) \right) \quad (91)$$

with the boundary conditions $\psi_x(0, t) = 0$ and $\psi_x(1, t) = 0$, where ψ_x is the deviation of the flux gradient from a reference flux gradient profile, μ_0 is the permeability of free space, $\eta(x)$ is the plasma resistivity and a is the radius of the last closed magnetic surface (LCMS). To obtain the finite-dimensional

state-space representation of the PDE, we discretize the PDE in the spatial domain $(0, 1)$. The state-space model is then

$$\dot{\psi}_x(t) = A(\eta(x)) \psi_x(t), \quad (92)$$

where $A(\eta(x)) \in \mathbb{R}^{N \times N}$ has the following non-zero entries.

$$a_{11} = \frac{-4}{3\mu_0 \Delta x^2 a^2} \left(\frac{\eta(x_{\frac{3}{2}})}{x_{\frac{3}{2}}} + \frac{2\eta(x_{\frac{5}{4}})}{x_{\frac{5}{4}}} \right),$$

$$a_{12} = \frac{4}{3\mu_0 \Delta x^2 a^2} \left(\frac{\eta(x_{\frac{3}{2}}) x_2}{x_{\frac{3}{2}}} \right), \quad (93)$$

$$a_{j,j-1} = \frac{1}{\Delta x^2 \mu_0 a^2} \left(\frac{\eta(x_{j-\frac{1}{2}})}{x_{j-\frac{1}{2}}} x_{j-1} \right) \text{ for } j = 2, \dots, N-1 \quad (94)$$

$$a_{j,j} = \frac{-1}{\Delta x^2 \mu_0 a^2} \left(\frac{\eta(x_{j+\frac{1}{2}})}{x_{j+\frac{1}{2}}} + \frac{\eta(x_{j-\frac{1}{2}})}{x_{j-\frac{1}{2}}} \right) x_j$$

for $j = 2, \dots, N-1$ (95)

$$a_{j,j+1} = \frac{1}{\Delta x^2 \mu_0 a^2} \left(\frac{\eta(x_{j+\frac{1}{2}})}{x_{j+\frac{1}{2}}} x_{j+1} \right) \text{ for } j = 2, \dots, N-1 \quad (96)$$

$$a_{N,N-1} = \frac{4}{3\Delta x \mu_0 a^2} \frac{\eta(x_{N-\frac{1}{2}}) x_{N-1}}{x_{N-\frac{1}{2}} \Delta x},$$

$$a_{N,N} = \frac{-4}{3\Delta x \mu_0 a^2} \left(\frac{2\eta(x_{N+\frac{1}{4}}) x_N}{x_{N+\frac{1}{4}} \Delta x} + \frac{\eta(x_{N-\frac{1}{2}}) x_N}{x_{N-\frac{1}{2}} \Delta x} \right), \quad (97)$$

where $\Delta x = \frac{1}{N}$ and $x_j := (j - \frac{1}{2})\Delta x$.

We discretize the model at $N = 7$ points. Typically the $\eta(x_k)$ are not precisely known (they depend on other state variables), so we substitute for $\eta(x_k)$ in (92) with $\hat{\eta}(x_k) + \alpha_j$, where $\hat{\eta}(x_k)$ are the nominal values of $\eta(x_k)$ and α_j are the uncertain parameters. At $x_k = 0.036, 0.143, 0.286, 0.429, 0.571, 0.714, 0.857, 0.964$, we use data from the Tore Supra reactor to estimate the $\hat{\eta}(x_k)$ as $1.775 \cdot 10^{-8}, 2.703 \cdot 10^{-8}, 5.676 \cdot 10^{-8}, 1.182 \cdot 10^{-7}, 2.058 \cdot 10^{-7}, 3.655 \cdot 10^{-7}, 1.076 \cdot 10^{-6}, 8.419 \cdot 10^{-6}$. The uncertain system is then written as

$$\dot{\psi}_x(t) = A(\alpha) \psi_x(t), \quad (98)$$

where A is affine, $A(\alpha) = A_0 + \sum_{i=1}^8 A_i \alpha_i$ (the A_i are omitted for the sake of brevity). For a given ρ , we restrict the uncertain parameters α_j to S_ρ , defined as

$$S_\rho := \{\alpha \in \mathbb{R}^8 : \sum_{i=1}^8 \alpha_i = -6|\rho|, -|\rho| \leq \alpha_i \leq |\rho|\}, \quad (99)$$

which is a simplex translated to the origin. We would like to determine the maximum value of ρ such that the system is stable by solving the following optimization problem.

$$\max \quad \rho$$

s.t. System (98) is stable for all $\alpha \in S_\rho$. (100)

To represent S_ρ using the standard unit simplex defined in (7), we define the invertible map $g : \Delta_8 \rightarrow S_\rho$ as

$$g(\alpha) = [g_1(\alpha) \quad \dots \quad g_8(\alpha)], \quad g_i(\alpha) := 2|\rho|(\alpha_i - 0.5). \quad (101)$$

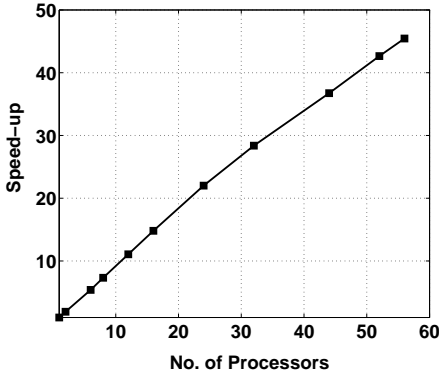


Fig. 7. Speed-up of set-up and SDP algorithms vs. number of processors for a discretized model of magnetic flux in Tokamak

Then, if we let $A'(\alpha) = A(g(\alpha))$, since g is one-to-one, $\{A(\alpha') : \alpha' \in S_\rho\} = \{A(g(\alpha)) : \alpha \in \Delta_8\} = \{A'(\alpha) : \alpha \in \Delta_8\}$. Thus stability of $\dot{\psi}_x(t) = A'(\alpha)\psi_x(t)$, for all $\alpha \in \Delta_l$ is equivalent to stability of Equation (98) for all $\alpha \in S_\rho$.

We solve the optimization problem in (100) using bisection. For each trial value of ρ , we use the proposed parallel SDP solver to solve the associated SDP obtained by the parallel set-up algorithm. The SDP problems have 224 constraints with the primal variable $X \in \mathbb{R}^{1092 \times 1092}$. The normalized maximum value of ρ is found to be 0.0019. In this particular example, the optimal value of ρ does not change with the degrees of $P(\alpha)$ and Polya's exponents d_1 and d_2 , primarily because the model is affine.

The SDPs are constructed and solved on a parallel Linux-based cluster Cosmea at Argonne National Laboratory. Fig. 7 shows the algorithm speed-up vs. the number of processors. Note that solving this problem by SOSTOOLS [21] on the same machine is impossible due to the lack of unallocated memory.

2) Example 2: Accuracy and Convergence

The goal of this example is to investigate the effect of the degree of $P(\alpha)$, d_p , and the Polya's exponents, d_1, d_2 on the accuracy of the algorithm. Given a computer with fixed amount of RAM, we compare the accuracy of the proposed algorithm with SOS algorithm. Consider the system $\dot{x}(t) = A(\alpha)x(t)$ where A is a polynomial degree 3 defined as

$$A(\alpha) = A_1\alpha_1^3 + A_2\alpha_1^2\alpha_2 + A_3\alpha_1\alpha_2\alpha_3 + A_4\alpha_1\alpha_3^2 + A_5\alpha_2^3 + A_6\alpha_3^3 \quad (102)$$

with the constraint

$$\alpha \in S_L := \left\{ \alpha \in \mathbb{R}^3 : \sum_{i=1}^3 \alpha_i = 2L + 1, L \leq \alpha_i \leq 1 \right\}$$

$$\begin{aligned} A_1 &= \begin{bmatrix} -0.61 & -0.56 & 0.402 \\ -0.48 & -0.550 & 0.671 \\ -1.01 & -0.918 & 0.029 \end{bmatrix}, & A_2 &= \begin{bmatrix} -0.484 & -0.86 & 1.5 \\ -0.732 & -0.841 & -0.126 \\ 0.685 & 0.305 & 0.106 \end{bmatrix} \\ A_3 &= \begin{bmatrix} -0.357 & 0.344 & -0.661 \\ -0.210 & -0.505 & 0.588 \\ 0.268 & 0.487 & -0.846 \end{bmatrix}, & A_4 &= \begin{bmatrix} -0.881 & -0.436 & 0.228 \\ 0.503 & -0.812 & 0.249 \\ -0.012 & 0.542 & -0.536 \end{bmatrix} \\ A_5 &= \begin{bmatrix} -0.703 & -0.298 & -0.178 \\ 0.402 & -0.761 & -0.300 \\ -0.010 & 0.461 & -0.588 \end{bmatrix}, & A_6 &= \begin{bmatrix} -0.201 & -0.182 & -0.557 \\ 0.803 & -0.412 & -0.203 \\ -0.440 & 0.011 & -0.881 \end{bmatrix} \end{aligned}$$

Defining g as in Example 1, the problem is

$$\begin{aligned} \min \quad & L \\ \text{s.t.} \quad & \dot{x}(t) = A(g(\alpha))x(t) \text{ is stable for all } \alpha \in \Delta_3. \end{aligned} \quad (103)$$

TABLE I
UPPER BOUNDS FOUND FOR L_{opt} BY SOS ALGORITHM USING DIFFERENT DEGREES FOR x AND α (INF: INFEASIBLE, O.M.: OUT OF MEMORY)

Degree in α		0	1	2
Degree in x	1	inf.	inf.	inf.
	2	inf.	-0.102	O.M.
	3	inf.	O.M.	O.M.

Using bisection in L , as in Example 1, we varied the parameters d_p , d_1 and d_2 . The cluster computer Karlin at the Illinois Institute of Technology with 24 Gbytes/node of RAM (216 Gbytes total memory) was used to run our algorithm. The upper bounds on the optimal L are shown in Fig. 8 in terms of d_1 and d_2 and for different d_p . Considering the optimal value of L to be $L_{opt} = -0.111$, Fig. 8 shows how increasing d_p and/or d_1, d_2 - when they are still relatively small - improves the accuracy of the algorithm. Fig. 9 demonstrates how the error in our upper bound for L_{opt} decreases by increasing d_p and/or d_1, d_2 .

For comparison, we solved the same stability problem using the SOS algorithm [21] using only a single node of the same cluster computer and 24 Gbytes of RAM. We used the Positivstellensatz approach based on [57] to impose the constraints $\sum_{i=1}^3 \alpha_i = 2L + 1$ and $L \leq \alpha_i \leq 1$. Table I shows the upper bounds on L given by the SOS algorithm using different degrees for x and α . By considering a Lyapunov function of degree two in x and degree one in α , the SOS algorithm gives -0.102 as the upper bound on L_{opt} as compared with our value of -0.111 . Increasing the degree of α in the Lyapunov function beyond degree two resulted in a failure due to lack of memory. Note that while relevant, this comparison may not be entirely fair as the SOS algorithm has not been decentralized and it can handle global nonlinear stability problems, which our algorithm cannot.

3) Example 3: Speed-up

In this example we evaluate the efficiency of the algorithm in using additional processors to decrease computation time. As mentioned in Section IV-D on computational complexity, the measure of this efficiency is termed speed-up and in Case 3, we gave a formula for this number. To evaluate the true speed-up, we first ran the set-up algorithm on the Blue Gene supercomputer at Argonne National Laboratory using three random linear systems with different state-space dimensions and numbers of uncertain parameters. Fig. 10 shows a log-log plot of the computation time of the set-up algorithm vs. the number of processors. As can be seen, the scalability of the algorithm is practically ideal for several different state-space dimensions and numbers of uncertain parameters.

To evaluate the speed-up of the SDP portion of the algorithm, we solved three random SDP problems with different dimensions using the Karlin cluster computer. Fig. 11 gives a log-log plot of the computation time of the SDP algorithm vs. the number of processors for three different dimensions of the primal variable X and the dual variable y . As indicated in the figure, the three dimensions of the primal variable X are 200, 385 and 1092, and the dimensions of the dual variable y are $K = 50, 90$ and 224, respectively. In all cases,

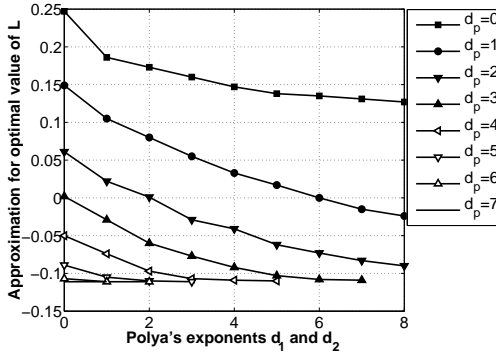


Fig. 8. Upper bound on optimal L vs. Polya's exponents d_1 and d_2 , for different degrees of $P(\alpha)$. ($d_1 = d_2$).

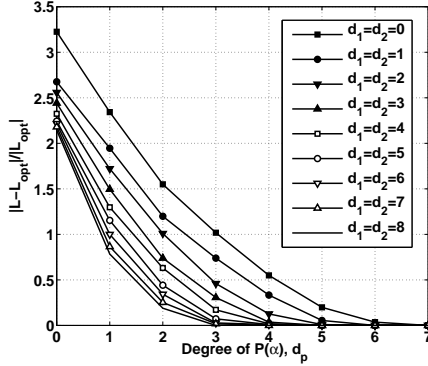


Fig. 9. Error of the approximation for the optimal value of L vs. degrees of $P(\alpha)$, for different Polya's exponents

$d_p = 2$ and $d_1 = d_2 = 1$. The linearity of the Time vs. Number of Processors curves in all three cases demonstrates the scalability of the SDP algorithm.

For comparison, we plot the speed-up of our algorithm vs. that of the general-purpose parallel SDP solver SDPARA 7.3.1 as illustrated in Fig. 12. Although similar for a small number of processors, for a larger number of processors, SDPARA saturates, while our algorithm remains approximately linear.

4) Example 4: Max state-space and parameter dimensions for a 9-node Linux cluster computer

The goal of this example is to show that given moderate computational resources, the proposed decentralized algorithms can solve robust stability problems for systems with 100+ states. We used the Karlin cluster computer with 24 Gbytes/node RAM and nine nodes. We ran the set-up and SDP algorithms to solve the robust stability problem with dimension n and l uncertain parameters on one and nine nodes of Karlin cluster computer. Thus the total memory access was thus 24 Gig and 216 Gig, respectively. Using trial and error, for different n and d_1, d_2 we found the largest l for which the algorithms do not terminate due to insufficient memory (Fig. 13). In all of the runs $d_a = d_p = 1$. Fig. 13 shows that by using 216 Gbytes of RAM, the algorithms can solve the stability problem of size $n = 100$ with 4 uncertain parameters in $d_1 = d_2 = 1$ Polya's iteration and with 3 uncertain parameters in $d_1 = d_2 = 4$ Polya's iterations.

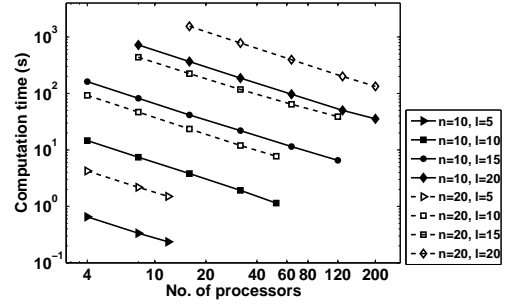


Fig. 10. Computation time of the parallel set-up algorithm vs. number of processors for different dimensions of linear system n and numbers of uncertain parameters l - executed on Blue Gene supercomputer of Argonne National Laboratory

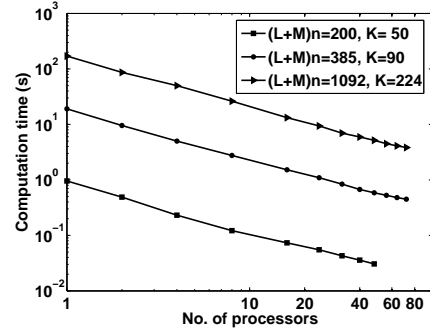


Fig. 11. Computation time of the parallel SDP algorithm vs. number of processors for different dimensions of primal variable $(L + M)n$ and of dual variable K - executed on Karlin cluster computer of Illinois Institute of Technology

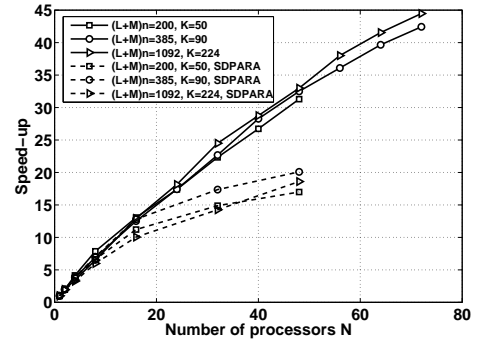


Fig. 12. Comparison between the speed-up of the present SDP solver and SDPARA 7.3.1, executed on Karlin cluster computer

VI. CONCLUSION

In this paper, we have presented a cluster-computing and supercomputing approach to stability analysis of large-scale linear systems of the form $\dot{x}(t) = A(\alpha)x(t)$ where A is polynomial, $\alpha \in \Delta_l \subset \mathbb{R}^l$ and $x \in \mathbb{R}^n$ and where $n \cong 100$ or $\alpha \cong 10$. The approach is based on mapping the structure of the LMI conditions associated with Polya's theorem to a decentralized computing environment. We have shown that for a sufficient number of processors, the proposed algorithm can solve the NP-hard robust stability problem with the same per-core computation cost as solving the Lyapunov inequality for a system with no parametric uncertainty. Theoretical and experimental results verify near-perfect scalability and speed-up for up to 200 processors. Moreover, numerical examples demonstrate the ability of the algorithm to perform robust

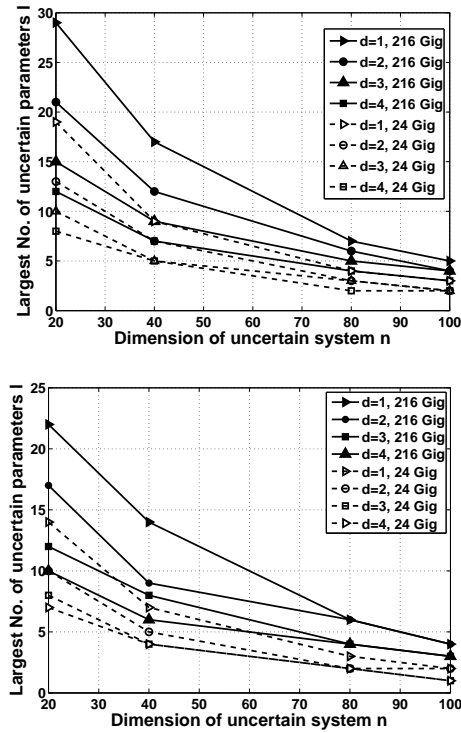


Fig. 13. Largest number of uncertain parameters of n -dimensional systems for which the set-up algorithm (Left) and SDP solver (Right) can solve the robust stability problem of the system using 24 and 216 Gg of RAM

analysis of systems with 100+ states and several uncertain parameters using a simple 9-node Linux cluster computer. We have also argued that our algorithms can also be extended to solve nonlinear stability analysis and robust controller synthesis problems, although this is left for future work.

REFERENCES

- [1] S. Gugercin and A. Antoulas, "A survey of model reduction by balanced truncation and some new results," *International Journal of Control*, vol. 77, no. 8, pp. 748–766, 2004.
- [2] J. Ackermann, A. Bartlett, D. Kaesbauer, W. Sienel, and R. Steinhauser, *Robust Control: Systems with Uncertain Physical Parameters*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.
- [3] S. P. Bhattacharyya, H. Chapellat, and L. H. Keel, *Robust Control: The Parametric Approach*. Prentice Hall, 1995.
- [4] M. Green and D. J. N. Limebeer, *Linear robust control*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.
- [5] K. Zhou and J. Doyle, *Essentials of Robust Control*. Prentice Hall, 1998.
- [6] V. Blondel and J. Tsitsiklis, "A survey of computational complexity results in systems and control," *Automatica*, vol. 36, no. 9, pp. 1249–1274, 2000.
- [7] A. Nemirovskii, "Several NP-hard problems arising in robust stability analysis," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 6, no. 2, pp. 99–105, 1993.
- [8] D. Walker and J. Dongarra, "Mpi: a standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.
- [9] A. Packard and J. Doyle, "Quadratic stability with real and complex perturbations," *IEEE Transactions on Automatic Control*, vol. 35, pp. 198–201, Feb 1990.
- [10] B. R. Barmish and C. L. DeMarco, "A new method for improvement of robustness bounds for linear state equations," in *Proceedings Conf. Inform. Sci. Syst. Princeton University*, 1986.
- [11] P. Gahinet, P. Apkarian, and M. Chilali, "Affine parameter-dependent lyapunov functions and real parametric uncertainty," *IEEE Transactions on Automatic Control*, vol. 41, pp. 436–442, Mar 1996.
- [12] R. C. L. F. Oliveira and P. L. D. Peres, "Stability of polytopes of matrices via affine parameter-dependent Lyapunov functions: Asymptotically exact LMI conditions," *Linear Algebra Appl.*, vol. 405, pp. 209–228, Aug 2005.
- [13] R. C. L. F. Oliveira and P. L. D. Peres, "A less conservative LMI condition for the robust stability of discrete-time uncertain systems," *Syst. Control Lett.*, vol. 43, pp. 371–378, Aug 2001.
- [14] D. Ramos and P. Peres, "An LMI approach to compute robust stability domains for uncertain linear systems," *Proceedings of the American Control Conference*, Jun 2001.
- [15] P. A. Bliman, "An existence result for polynomial solutions of parameter dependent LMIs," *Systems & Control Letters*, no. 3-4, pp. 165–169, 2004.
- [16] M. Peet, "Exponentially stable nonlinear systems have polynomial lyapunov functions on bounded regions," *Automatic Control, IEEE Transactions on*, vol. 54, no. 5, pp. 979–987, 2009.
- [17] A. Ben-Tal and A. Nemirovski, "Robust convex optimization," *Math. Operat. Res.*, vol. 23, no. 4, pp. 769–805, 1998.
- [18] P. A. Bliman, "A convex approach to robust stability for linear systems with uncertain scalar parameters," *SIAM J. Control Optim.*, vol. 42, no. 3-4, pp. 2016–2042, 2004.
- [19] X. Zhang and P. Tsiotras, "Parameter-dependent lyapunov functions for stability analysis of LTI parameter dependent systems," pp. 5168–5173, in *Proceedings of the IEEE 42nd Conference on Decision and Control*, 2003.
- [20] X. Zhang, P. Tsiotras, and P. A. Bliman, "Multi-parameter dependent lyapunov functions for the stability analysis of parameter-dependent LTI systems," pp. 1263–1268, in *Proceedings of IEEE International Symposium on, Mediterrean Conference on Control and Automation*, 2005.
- [21] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, "Introducing SOS-TOOLS: a general purpose sum of squares programming solver," *Proceedings of IEEE Conference on Decision and Control*, 2002.
- [22] D. Henrion and J. B. Lasserre, "Gloptipoly: Global optimization over polynomials with Matlab and SeDuMi," *Proceedings of IEEE Conference on Decision and Control*, Mar 2003.
- [23] C. W. Scherer and C. W. J. Hol, "Matrix sum-of squares relaxations for robust semi-definite programs," *Math. programming Ser. B*, vol. 107, no. 1-2, pp. 189–211, 2006.
- [24] G. Chesi, A. Garulli, A. Tesi, and A. Vicino, "Polynomially parameter-dependent lyapunov functions for robust stability of polytopic systems: an LMI approach," *IEEE Transactions on Automatic Control*, vol. 50, pp. 365–370, Mar 2005.
- [25] R. C. L. F. Oliveira and P. L. D. Peres, "Parameter-dependent LMIs in robust analysis: Characterization of homogeneous polynomially parameter-dependent solutions via LMI relaxations," *IEEE Transactions on Automatic Control*, vol. 52, pp. 1334–1340, Jul 2007.
- [26] R. C. L. F. Oliveira, P. A. Bliman, and P. L. D. Peres, "Robust LMIs with parameters in multi-simplex: Existence of solutions and applications," pp. 2226–2231, *Proceedings of IEEE Conference on Decision and Control*, 2008.
- [27] B. Borchers and J. G. Young, "Implementation of a primal dual method for SDP on a shared memory parallel architecture," *Computational Optimization and Applications*, vol. 37, no. 3, pp. 355–369, 2007.
- [28] M. Yamashita, K. Fujisawa, and M. Kojima, "SDPARA: Semidefinite programming algorithm parallel version," *Parallel Computing*, vol. 29, pp. 1053–1067, 2003.
- [29] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," No. 30, pp. 483–485, *AFIPS Conference Proceedings*, 1967.
- [30] L. Kalé, B. Ramkumar, A. Sinha, and A. Gursay, "The charm parallel programming language and system: Part i-description of language features," *Parallel Programming Laboratory Technical Report No. 95-02*, 1994.
- [31] S. Deitz, *High-level programming language abstractions for advanced and dynamic parallel computations*. PhD thesis, Computer Science and Engineering Department, University of Washington, 2005.
- [32] K. Gatermann and P. Parrilo, "Symmetry groups, semidefinite programs, and sums of squares," *Journal of Pure and Applied Algebra*, vol. 192, no. 1, pp. 95–128, 2004.
- [33] P. Parrilo, "Exploiting algebraic structure in sum of squares programs," *Positive polynomials in control*, pp. 580–580, 2005.
- [34] S. Kim, M. Kojima, and H. Waki, "Generalized lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems," *SIAM Journal on Optimization*, vol. 15, no. 3, pp. 697–719, 2005.

- [35] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto, "Algorithm 883: Sparsepop—a sparse semidefinite programming relaxation of polynomial optimization problems," *ACM Trans. Math. Softw.*, vol. 35, no. 2, 2008.
- [36] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*, vol. 10. Springer Verlag, 2007.
- [37] B. Buchberger and F. Winkler, *Gröbner bases and applications*, vol. 251. Cambridge Univ Pr, 1998.
- [38] M. M. Peet and Y. V. Peet, "A parallel-computing solution for optimization of polynomials," Proceedings of the American Control Conference, Jun-Jul 2010.
- [39] E. Scheinerman, *Mathematics: A Discrete Introduction, Second Edition*. Thomson Brooks/Cole Publishing Co., United States, 2005.
- [40] G. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*. Cambridge University Press, 1934.
- [41] M. Castle, V. Powers, and B. Reznick, "A quantitative polya's theorem with zeros," *Effective Methods in Algebraic Geometry*, vol. 44, no. 9, pp. 1285–1290, 2009.
- [42] P. Gahinet and P. Apkarian, "A linear matrix inequality approach to H infinity control," *International Journal of Robust and Nonlinear Control*, vol. 4, pp. 421–448, 1994.
- [43] G. Dullerud and F. Paganini, *A course in robust control theory*, vol. 6. Springer New York, 2000.
- [44] S. J. Benson, "DSDP3: Dual scaling algorithm for general positive semidefinite programs," *Technical Report. ANL/MCS-P851-1000, Argonne National Labs*, 2001.
- [45] S. J. Benson, Y. Ye, and X. Zhang, "Solving large-scale sparse semidefinite programs for combinatorial optimization," *SIAM Journal on Optimization*, vol. 10, pp. 443–461, 1998.
- [46] F. Alizadeh, J. A. Haeberly, and M. Overton, "Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 746–768, 1998.
- [47] R. D. C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM Journal on Optimization*, vol. 7, no. 3, 1997.
- [48] C. Helmberg, F. R. R. J. Vanderbei, and H. Wolkovitz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, vol. 6, pp. 342–361, 1996.
- [49] C. Helmberg and F. Rendl, "A spectral bundle method for semidefinite programming," *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 673–696, 2000.
- [50] K. K. Sivaramakrishnan, "A parallel interior point decomposition algorithm for block angular semidefinite programs," *Comput. Optim. Appl.*, vol. 46, no. 1, pp. 1–29, 2010.
- [51] M. Nayakkankuppam, "Solving large-scale semidefinite programs in parallel," *Mathematical programming*, vol. 109, no. 2, pp. 477–504, 2007.
- [52] M. L. O. F. Alizadeh, J. P. A. Haeberly, "Primal-dual interior-point methods for semidefinite programming," *Math Programming Symposium*, Ann Arbor 1994.
- [53] F. Alizadeh, J. Haeberly, and M. Overton, "Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 746–768, 1998.
- [54] R. Greenlaw, H. Hoover, and W. Ruzzo, *Limits to parallel computation: P-completeness theory*. Oxford University Press, USA, 1995.
- [55] J. Sturm, "Using sedumi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11–12, pp. 625–653, 1999.
- [56] E. Witrant, E. Joffrin, S. Brémont, G. Giruzzi, D. Mazon, O. Barana, and P. Moreau, "A control-oriented model of the current profile in tokamak plasma," *Plasma Physics and Controlled Fusion*, vol. 49, pp. 1075–1105, 2007.
- [57] G. Stengle, "A nullstellensatz and a positivstellensatz in semialgebraic geometry," *Mathematische Annalen*, vol. 207, no. 2, pp. 87–97, 1973.



of stability and control of large-scale complex systems.

Reza Kamyar received the B.S. and M.S. in aerospace engineering from Sharif University of Technology, Tehran, Iran in 2008, and 2010. He is currently a Ph.D student in the department of mechanical engineering of Arizona State University, Tempe, Arizona. He is a research assistant with Cybernetic Systems and Controls Laboratory (CSCL) in the School for Engineering of Matter, Transport and Energy (SEMTE) at Arizona State University. His research focuses on the development of decentralized algorithms applied to the problems



and Aerospace Engineering Department, Illinois Institute of Technology, Chicago, from 2008 to 2012. He is currently an Assistant Professor of Aerospace Engineering in the School for the Engineering of Matter, Transport, and Energy at Arizona State University, Tempe, and Director of the Cybernetic Systems and Controls Laboratory. His research interests are in the role of computation as it is applied to the understanding and control of complex and large-scale systems. Applications include fusion energy and immunology. Dr. Peet received an NSF CAREER award in 2011.

Matthew M. Peet received the B.S. degrees in physics and in aerospace engineering from the University of Texas at Austin in 1999 and the M.S. and Ph.D. degrees in aeronautics and astronautics from Stanford University, Stanford, CA, in 2001 and 2006, respectively. He was a Postdoctoral Fellow at the National Institute for Research in Computer Science and Control (INRIA), Paris, France, from 2006 to 2008, where he worked in the SISYPHE and BANG groups. He was an Assistant Professor of Aerospace Engineering in the Mechanical, Materials,



NSF research and teaching fellow at Northwestern University and assistant computational scientist at the Mathematics and Computer Science Division at Argonne National Laboratory in 2009–2012.

Yulia Peet is an Assistant Professor of Mechanical and Aerospace Engineering at the School for Engineering of Matter, Transport and Energy at Arizona State University. Her Ph.D. degree is in Aeronautics and Astronautics from Stanford (2006), M.S. in Aerospace Engineering (1999) and B.S. in Applied Mathematics and Physics (1997) from Moscow Institute of Physics and Technology in Russia. Her previous appointments include a postdoctoral position at the University of Pierre and Marie Curie in Paris in 2006–2008, and a dual appointment as an